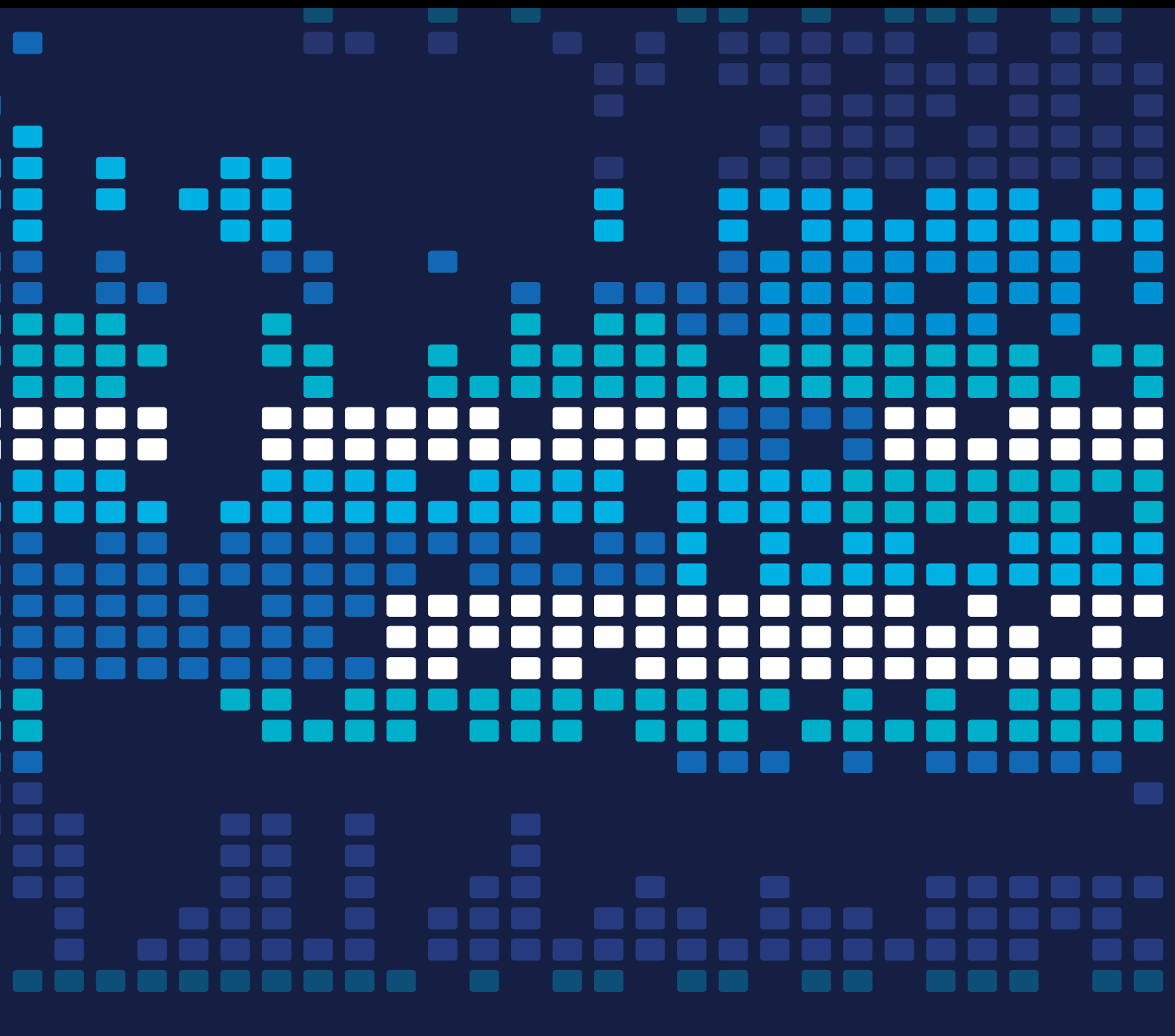
Scientific Programming

Resource Management in Virtualized Clouds

Guest Editors: Ligang He, Laurence T. Yang, Zhihui Du, and Florin Pop



Resource Management in Virtualized Clouds

Scientific Programming

Resource Management in Virtualized Clouds

Guest Editors: Ligang He, Laurence T. Yang, Zhihui Du, and Florin Pop

Editorial Board

Contents

Resource Management in Virtualized Clouds
Ligang He, Laurence T. Yang, Zhihui Du, and Florin Pop
Volume 2016, Article ID 1407940, 2 pages

Revisiting the Practicality of Search on Encrypted Data: From the Security Broker's Perspective
Peiyi Han, Chuanyi Liu, Binxing Fang, Guofeng Wang, Xiaobao Song, and Lei Wan
Volume 2016, Article ID 8057208, 9 pages

Distributed Parallel Endmember Extraction of Hyperspectral Data Based on Spark
Zebin Wu, Jinping Gu, Yonglong Li, Fu Xiao, Jin Sun, and Zhihui Wei
Volume 2016, Article ID 3252148, 9 pages

A Two-Tier Energy-Aware Resource Management for Virtualized Cloud Computing System
Wei Huang, Zhen Wang, Mianxiong Dong, and Zhuzhong Qian
Volume 2016, Article ID 4386362, 15 pages

Biobjective VoIP Service Management in Cloud Infrastructure
Jorge M. Cortés-Mendoza, Andrei Tchernykh, Fermin A. Armenta-Cano, Pascal Bouvry, Alexander Yu. Drozdov, and Loic Didelot
Volume 2016, Article ID 5706790, 14 pages

Task Classification Based Energy-Aware Consolidation in Clouds
HeeSeok Choi, JongBeom Lim, Heonchang Yu, and EunYoung Lee
Volume 2016, Article ID 6208358, 13 pages

Optimized Virtual Machine Placement with Traffic-Aware Balancing in Data Center Networks
Tao Chen, Xiaofeng Gao, and Guihai Chen
Volume 2016, Article ID 3101658, 10 pages

MultiCache: Multilayered Cache Implementation for I/O Virtualization
Jaechun No and Sung-soon Park
Volume 2016, Article ID 3780163, 13 pages

VR-Cluster: Dynamic Migration for Resource Fragmentation Problem in Virtual Router Platform
Xianming Gao, Baosheng Wang, and Xiaozhe Zhang
Volume 2016, Article ID 3976965, 14 pages

FP-ABC: Fast and Parallel ABC Based Energy-Efficiency Live VM Allocation Policy in Data Centers
Jianhua Jiang, Yunzhao Feng, Milan Parmar, and Keqin Li
Volume 2016, Article ID 9524379, 9 pages

A User-Customized Virtual Network Platform for NaaS Cloud
Lei Xiao, Yu Sheng, Guanlan Tan, Jianxin Wang, and Yi Pan
Volume 2016, Article ID 9315672, 6 pages

FPGA-Aware Scheduling Strategies at Hypervisor Level in Cloud Environments
Julio Proaño Orellana, Blanca Caminero, Carmen Carrión, Luis Tomas, Selome Kostentinos Tesfatsion, and Johan Tordsson
Volume 2016, Article ID 4670271, 12 pages

ANCS: Achieving QoS through Dynamic Allocation of Network Resources in Virtualized Clouds
Cheol-Ho Hong, Kyungwoon Lee, Hyunchan Park, and Chuck Yoo
Volume 2016, Article ID 4708195, 10 pages

Mining Software Repositories for Automatic Interface Recommendation
Xiaobing Sun, Bin Li, Yucong Duan, Wei Shi, and Xiangyue Liu
Volume 2016, Article ID 5475964, 11 pages

On Elasticity Measurement in Cloud Computing
Wei Ai, Kenli Li, Shenglin Lan, Fan Zhang, Jing Mei, Keqin Li, and Rajkumar Buyya
Volume 2016, Article ID 7519507, 13 pages

Optimizing Checkpoint Restart with Data Deduplication
Zhengyu Chen, Jianhua Sun, and Hao Chen
Volume 2016, Article ID 9315493, 11 pages

Research on Linux Trusted Boot Method Based on Reverse Integrity Verification
Chenlin Huang, Chuanwang Hou, Huadong Dai, Yan Ding, Songling Fu, and Mengluo Ji
Volume 2016, Article ID 4516596, 12 pages

Data Sets Replicas Placements Strategy from Cost-Effective View in the Cloud
Xiuguo Wu
Volume 2016, Article ID 1496714, 13 pages

A Randomization Approach for Stochastic Workflow Scheduling in Clouds
Wei Zheng, Chen Wang, and Dongzhan Zhang
Volume 2016, Article ID 9136107, 13 pages

A Heuristic Task Scheduling Algorithm for Heterogeneous Virtual Clusters
Weiwei Lin, Wentai Wu, and James Z. Wang
Volume 2016, Article ID 7040276, 10 pages

MHDFS: A Memory-Based Hadoop Framework for Large Data Storage
Aibo Song, Maoxian Zhao, Yingying Xue, and Junzhou Luo
Volume 2016, Article ID 1808396, 12 pages

Feedback-Based Resource Allocation in MapReduce-Based Systems
Bunjamin Memishi, María S. Pérez, and Gabriel Antoniu
Volume 2016, Article ID 7241928, 13 pages

Virtual Machine Placement Algorithm for Both Energy-Awareness and SLA Violation Reduction in Cloud Data Centers
Zhou Zhou, Zhigang Hu, and Keqin Li
Volume 2016, Article ID 5612039, 11 pages

Energy-Efficient Reliability-Aware Scheduling Algorithm on Heterogeneous Systems
Xiaoyong Tang and Weizhen Tan
Volume 2016, Article ID 9823213, 13 pages

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 1407940, 2 pages
http://dx.doi.org/10.1155/2016/1407940

Editorial

Resource Management in Virtualized Clouds

Ligang He,[1] Laurence T. Yang,[2] Zhihui Du,[3] and Florin Pop[4]

[1]*University of Warwick, Coventry, UK*
[2]*St. Francis Xavier University, Antigonish, NS, Canada*
[3]*Tsinghua University, Beijing, China*
[4]*University Politehnica of Bucharest, Bucharest, Romania*

Correspondence should be addressed to Ligang He; liganghe@dcs.warwick.ac.uk

Received 10 October 2016; Accepted 11 October 2016

Resource management is of paramount importance in achieving high performance in Cloud environments. Different from traditional parallel and distributed systems, resource virtualization is a key feature in Cloud systems. Although researchers have been developing various strategies and techniques to address the resource management issues in virtualized Clouds, they, as this special issue shows, still face many new research challenges.

This special issue aims to report the latest scientific advances in resource management techniques for virtualized Clouds, especially on the topics including VM consolidation strategies, scheduling techniques, techniques for managing various types of virtualized resource, and VM management for various types of application. This issue received 47 high quality submissions. After the rigorous review process, 23 papers were accepted in this issue, documenting the relevant research from China, Spain, USA, Australia, Korea, Mexico, Luxembourg, Russia, France, and so forth.

The research presented in these 23 papers broadly covers the interesting scope of this special issue. Among these papers, scheduling strategies are proposed for various types of application and platform. W. Zheng et al. from Xiamen University in China proposed a randomization approach for scheduling stochastic workflows. J. M. Cortés-Mendoza et al. from CICESE Research Center in Mexico developed a new method for scheduling VoIP (Voice over Internet Protocol) tasks in Clouds. W. Lin et al. from South China University of Technology in China designed a task scheduling algorithm for heterogeneous virtual clusters. J. P. Orellana et al. from University of Castilla-La Mancha in Spain investigated scheduling strategies for FPGA devices equipped in Clouds.

Different resource management techniques are also investigated for virtualized Clouds. For example, a novel task-classification-based VM consolidation method was proposed by H. Choi et al. from Korea University in Korea. T. Chen et al. from Shanghai Jiao Tong University in China developed a new VM placement method by taking traffic balancing into account.

In addition to managing computation resources, new techniques are investigated for managing other types of resource. As an example, the dynamic allocation of network resources was studied by C.-H. Hong et al. from Korea University in Korea. J. No et al. at Sejong University in Korea implemented the cache management scheme for virtualized I/O resources.

Finally, although virtual machine is still a popular medium for providing the running environments for tasks and services, other types of virtualization technologies are emerging. B. Memishi et al. from Universidad Politécnica de Madrid in Spain investigated the resource allocation based on containers, a fine-grain alternative virtualization technology to virtual machines. W. Ai et al. from Hunan University in China proposed a novel metric to measure the elasticity of virtualized resources. For this special issue, we are able to select excellent papers providing a range of methods on the theme of the special issue.

Acknowledgments

We would like to thank the authors for contributing to our special issue. We also thank the reviewers for taking their

valuable time to review and provide constructive comments to the authors.

Ligang He
Laurence T. Yang
Zhihui Du
Florin Pop

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 8057208, 9 pages
http://dx.doi.org/10.1155/2016/8057208

Research Article

Revisiting the Practicality of Search on Encrypted Data: From the Security Broker's Perspective

Peiyi Han,[1,2] Chuanyi Liu,[1] Binxing Fang,[1,3] Guofeng Wang,[1,2] Xiaobao Song,[4] and Lei Wan[4]

[1] *Harbin Institute of Technology (Shenzhen), Shenzhen 518055, China*
[2] *School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[3] *Institute of Electronic and Information Engineering in Dongguan, University of Electronic Science and Technology of China, Dongguan 523000, China*
[4] *Shenzhen Yunanbao Technology Co., Ltd., Shenzhen 518057, China*

Correspondence should be addressed to Chuanyi Liu; cy-liu04@mails.tsinghua.edu.cn

Received 28 February 2016; Accepted 14 September 2016

Academic Editor: Ligang He

The primary business challenge for the customers to use outsourced computation and storage is the loss of data control and security. So encryption will become a commodity in the near future. There is big diffusion with the above scenario: take advantage of current application's full functionalities at the same time ensuring their sensitive data remains protected and under customers' control. Prior works have achieved effective progress towards satisfying both sides. But there are still some technical challenges, such as supporting file or data-stream based applications and supporting full-text and advanced searches. In this paper, a novel security broker based encrypted data search scheme, called Enc-YUN, is proposed, which transparently builds a reverse index at the security broker when the data flow is transmitted to the cloud. And search firstly takes place on the index, in which the mapping structure corresponds to and retrieves the very encrypted data in the cloud on behalf of the client. With this scheme, updated-to-date full-text search techniques can be easily integrated to carry out the most advanced search functionalities, at the same time, maintaining the strongest levels of data protection from curious providers or third parties. Experimental results show that Enc-YUN is effective with broad categories of cloud applications, and the performance overhead induced is minor and acceptable according to user's perceptual experience.

1. Introduction

Specialization and outsourcing make society more efficient and scalable, and computing is not any different. According to Cisco global mobile data traffic report [1], cloud applications account for 83% of total mobile data traffic by 2015 and will account for 90% by 2019. The primary business challenge for the customers to use outsourced computation and storage is the loss of data control and security, especially with mission-critical applications or privacy-sensitive applications.

A promising solution is encryption, providing only encrypted data to the cloud. However, there exists diffusion: take advantage of current application's full functionalities at the same time ensuring their sensitive data remains protected and under customers' control. Take data search as an example; can customers still search the contents based on encrypted data?

Against the above problem, current efforts can be summarized into two categories: the first approach focuses on the Searchable Encryption (SE) Algorithms [2], which allow the data owner to delegate search capabilities to the cloud provider without decrypting the documents. However, this approach should modify the legacy cloud provider's interface to adopt the very SE library. And the search capabilities are limited to keyword granularity. The second approach often uses a proxy, namely, data security broker, which transparently sits between the cloud application and its users, intercepting critical data before it is passed into the cloud and replacing it with a random token or encryption value that is meaningless for the cloud.

This paper proposes security broker based architecture to protect the data privacy and search on encrypted data, called Enc-YUN, which builds a reverse index [3] to map the data transmitted and data in the cloud, and search firstly takes place on the index and then retrieves the corresponding encrypted data in the cloud on behalf of the client. With this scheme, updated-to-date full-text search techniques can be easily integrated to carry out the most advanced search functionalities, at the same time, maintaining the strongest levels of data protection from curious providers or third parties.

There is no free lunch, though. With the above scheme, some technical challenges are still waiting to be solved, which can be summarized as follows.

No Modifications on the Legacy Applications. Actually, rewriting or modifying applications in order to implement the Searchable Data Encryption Algorithms is always impossible for the application providers. Broker is located between the application and the user, which can intercept the information to the plaintext of the user before it is transferred to the applications and convert the ciphertext into plaintext before the information is sent to the user. The user should trust broker which is deployed on the user's premises. The broker and sensitive data are in the control of user, even though broker works by intercepting user data to cloud service. In fact, the broker provides an ability to adapt various cloud services transparently without modifying or even sacrificing usability.

Supporting More Categories of Applications While Being Non-Custom-Crafted. With Enc-YUN, it is necessary to recognize the protocol of the application to determine whether the application requires encrypted and semantic analysis on the content of protocol for obtaining the positions of sensitive data which need to be encrypted. However, lots of various SaaS applications need to be analyzed which is a big challenge! Enc-YUN should support popular SaaS applications as much as possible but should not analyze protocol of applications one by one. Then, we classified SaaS applications and had a protocol analysis about typical application in every category. If a new application needs to be protected, Enc-YUN would find the corresponding category and only require changing fewer codes. In the future, we will investigate mechanisms that fully automate analysis about protocol of application and semantic content of protocol.

Selective and Searchable Data Encryption. First, if we would encrypt simply all the content of protocol, this leads to the server-side of application parse data error and possibly denial of service. And the format of SaaS applications data such as phone number and email needs to be verified by cloud services. Thus, these data also should not be encrypted in case of impacting the functionality of SaaS applications. Second, keyword search is a common operation in SaaS applications, but it is often impractical to run on the client because it would require downloading large amounts of data to the user's machine. While there exist practical cryptographic schemes for keyword search, they require that cloud provider modify or rewrite the application code and interface.

The challenge facing Enc-YUN is how to implement selective and searchable data encryption. Enc-YUN could determine what data should be encrypted by policies related to the attribute of data. That greatly preserves the usability and user experience of cloud services.

2. Related Work

In this section, previous work has been well systematized and summarized into two parts. The first part is mainly about Searchable Data Encryption Algorithms by which user can search documents without decrypting them. Client controlled search on encrypted data is lucidly elaborated in the second part of this section. This approach is often implemented by a proxy called data security broker to intercept critical data before it is passed into the cloud and replace it with a random token or encryption value.

2.1. Searchable Data Encryption Algorithms. Traditional ciphertext search technologies can be summarized into two typical categories.

Linear search compares the words of ciphertext in turn to confirm whether or not the keyword exists and count the frequency of this keyword. For instance, Song et al. [4] proposed a solution based on searchable symmetric-key encryption (SSKE) scheme which adopts stream cipher method to encrypt character data.

Security index-based ciphertext search establishes a keyword index [5] according to the document and then encrypts and uploads index and document to the cloud. And keywords will be compared from the index instead of the whole document. Based on this approach, Boneh et al. [6] provided a method, namely, public-key encryption with keyword search (PEKS) [7, 8], such that the recipient searches keywords from the file sent by the sender. And Agrawal et al. [9] in IBM Research Center proposed an order preserving encryption scheme (OPES) algorithm [10] that keeps the values in order during the encryption process.

However, the conventional ciphertext search methods proposed above are based on the premise that the cloud provider needs to change the interface of existing cloud services. In fact, it is difficult for providers to do this in practice.

2.2. Client Controlled Search on Encrypted Data. The broker is the intermediary between cloud providers and users that encrypt and protect sensitive data of users. Consider several deployment locations of broker based on a simplified architecture of typical SaaS applications, outlined in Figure 1.

2.2.1. Between the Application's Client-Side and User. The broker can encrypt data at the point (Figure 1(a)) before the application code (including the client-side code) can access it. The application can only view an encrypted version of the data.

He et al. proposed a general solution which is ShadowCrypt [11] for encrypting textual data for existing web applications. ShadowCrypt runs as a browser extension, replacing

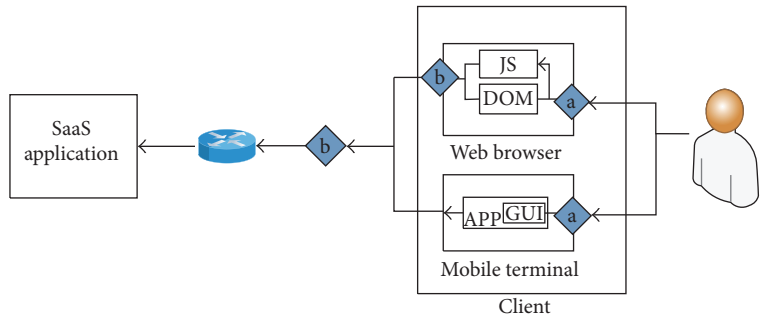


Figure 1: Architecture of typical SaaS applications and chokepoints for data encryption.

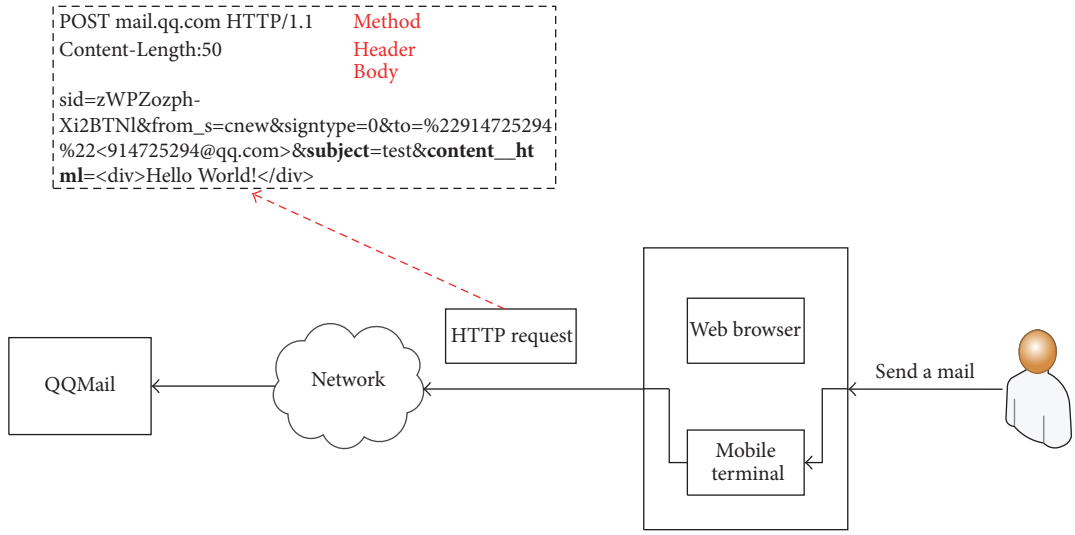


Figure 2: The procedure of sending mails to QQMail.

input elements in a page with secure, isolated shadow inputs and encrypted text with secure, isolated clear text. According to Figure 2, ShadowCrypt cannot encrypt data in mobile application. And Figure 3 shows file data which is uploaded to Dropbox by users is not stored in DOM nodes. Thus, ShadowCrypt could not support encrypting file data. In addition, it only applies to several web browsers in PC platforms such as Chrome and Firefox.

Mimesis Aegis [12] which is suitable for mobile platforms not only provides isolation but also preserves the user experience through the creation of a conceptual layer called Layer 7.5 (L-7.5), which is interposed between the application (OSI Layer 7) and the user (Layer 8). However, Mimesis could not support encrypting file data.

2.2.2. Between the Application's Client-Side and Network. The broker also can be deployed at the point (Figure 1(b)) to encrypt data after the application's client-side (i.e., JavaScript/HTML) would send data to the cloud services. Therefore, we can adopt the extension of browser and proxy as the broker.

Virtru [13] offers a browser plugin that performs email encryption, such that web-mail providers like Gmail cannot see users' data in the clear. But Virtru provides only a point solution for a handful of web-mail providers and does not generalize to other web applications and mobile applications.

Mylar [14] is an extension of the Meteor JavaScript framework for building applications that encrypt all their data sent to the server. Developers need to write their applications in Meteor (affecting backwards compatibility) and tell Mylar what data needs encryption.

3. Enc-YUN: Security Broker Based Search on Encrypted Data

There are three different parties in Enc-YUN: the users, the security broker, and the cloud services. Enc-YUN aims to protect the users' confidential data from attacking by hackers or intercepting by cloud providers.

3.1. Architecture. The architecture of Enc-YUN is shown in Figure 5. Enc-YUN consists of the five following components.

Parser. It intercepts data sent to and from the server, and it is responsible for recognizing application protocol and analyzing semantic content of protocol between the user and the

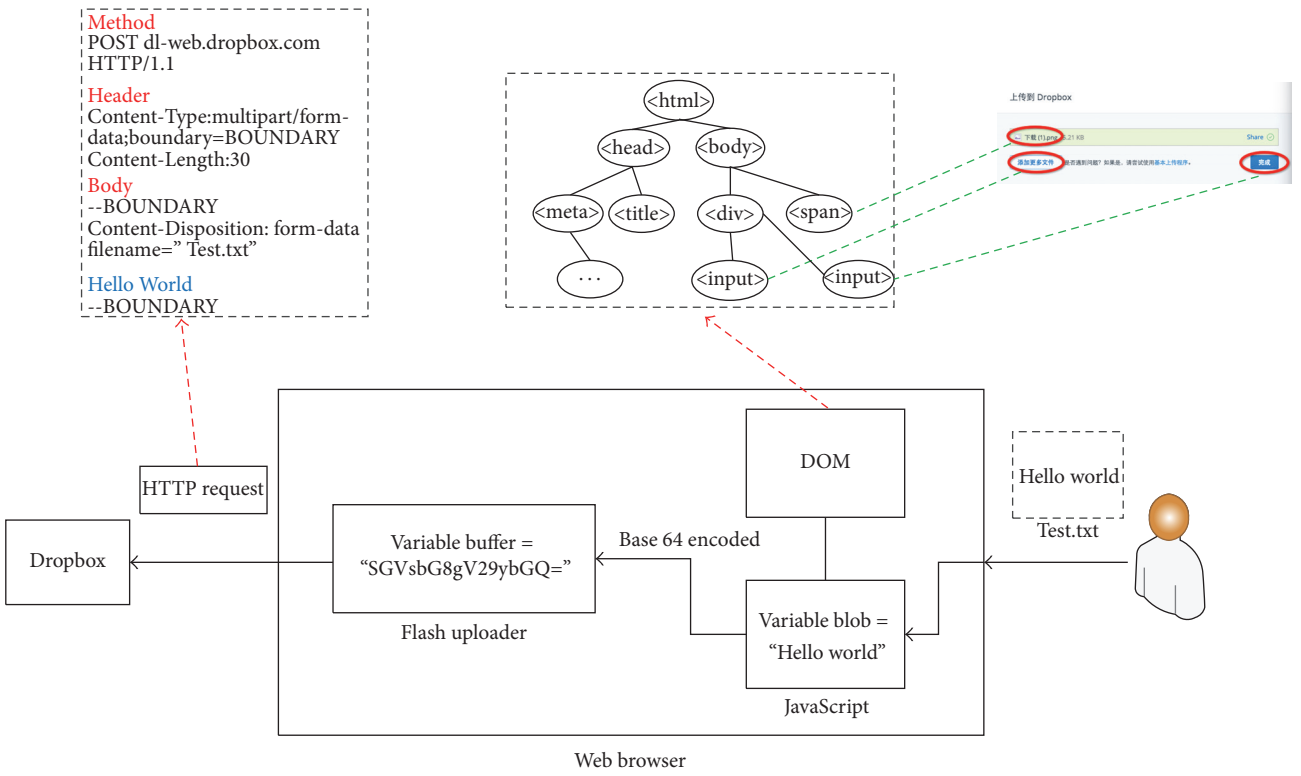


Figure 3: The procedure of uploading files to Dropbox.

cloud service in order to obtain what data require to be encrypted or decrypted. It supports multiple application protocols and various data formats. According to Figure 4, the broker could also intercept a secure channel such as SSL/TLS between users and cloud servers. The client sets SSL connection with the broker, and the broker establishes another SSL connection with the cloud server. As the broker works on behalf of clients, it is trusted by clients even when it decrypts the original data sent by clients.

Encrypter. It supports selective and searchable data encryption and integrates with several encryption algorithms such as AES and DES. Encrypter calls for secret keys from Key Manager before encrypting or decrypting and protects user sensitive data with distributed keys.

Searcher. It would transform metadata to MetaData Manager for ciphertext search. Searcher receives search keywords from users and searches content from cache which stores confidential data and return the results to users.

Transmitter. It is responsible for transmitting encrypted data from users to cloud services and data from SaaS applications to users.

Key Manager. It performs generation, storage, and management on keys used to encrypt or decrypt.

MetaData Manager. It stores and manages metadata.

3.2. Application Analysis. Enc-YUN not only supports popular SaaS applications as much as possible but also does not analyze protocol of applications one by one. In regard of the protocol recognition and semantic analysis when applying SaaS, we encountered two challenges:

(i) How to achieve the automatic adaptation of new application protocol if there are various protocols?

(ii) How to achieve the match of protocols without changing broker when the protocol changes?

The application of SaaS can be divided into the following categories: email, cloud storage, CRM, ERP, office 365, social category and so on. Analyzing the protocol of two typical applications of each category, we can find that the basic protocol is the same.

Figure 2 presents the content of the protocol about sending mail. Enc-YUN could obtain attributes and content of data by analyzing this typical format of key-value. For example, "subject" means mail subject and "content_html" means mail content. Then Enc-YUN encrypts the content of the subject and body of mail. Furthermore, Figure 3 shows the cloud storage applications adopt uploading content of files in multipart format. And parser parses the content with finite-state machine, encrypts data, assembles it into a new standard multipart format, and then forwards the request to the application. So we achieved the protocol adaptation and semantic identification of typical applications according to its category. If adding the new application is necessary,

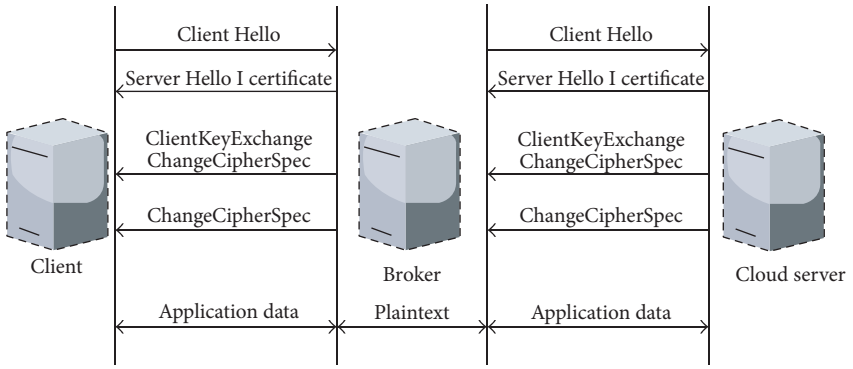


Figure 4: SSL/TLS interception in Enc-YUN.

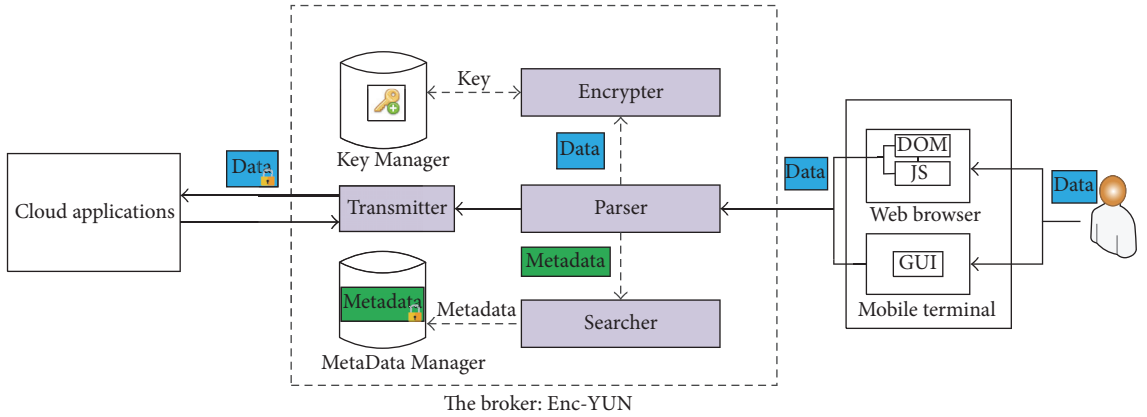


Figure 5: Architecture of the Enc-YUN.

we only need to find the category the application belongs to and achieve the protection of sensitive data in the new applications with little code modification.

Meanwhile, we will maintain an updated application feature library. Table 1 shows several features of QQMail in the library. If the protocol of the application changed, the application feature library will update the protocol feature and rules of semantic analysis of the application. And Enc-YUN could obtain new positions of sensitive data, extract data, and encrypt data by synchronizing the feature library.

3.3. Selective and Searchable Data Encryption. Cloud services would return error message when receiving request data which is encrypted or has incorrect format from users. And traditional keyword search is invalid on encrypted data. Selective and searchable data encryption [15] is another challenge for Enc-YUN.

Users have the right to choose if they want the data to be encrypted and can set some policies to control what data should be encrypted and what data should keep clear. Enc-YUN carries out selective encryption based on policies associated with attributes of confidential data.

Broker is deployed in the internal network of enterprise or organization controlled by users. Getting control of encryption keys to sensitive data, permission of selective data encryption updating policy, and metadata access, the broker is credible for the users. Under Enc-YUN, users get the control of sensitive data stored in SaaS application.

Then the paper proposed an approach of ciphertext search based on broker. The architecture of ciphertext search is shown in Figure 6:

(1) The user makes a keyword search request to the cloud application.

(2) Broker would intercept this request and receive the keyword from user. Then, the search query is executed against the local index.

(3) The local index, which stores a reverse index to map the keywords and data in the cloud, returns all of the associated metadata to the broker.

(4) The broker forwards the result to the user.

(5) The user retrieves the encrypted data or records according to the metadata ID which is the identity of encrypted data in cloud applications.

(6) Cloud applications return encrypted data which contains the keyword to the broker.

(7) Broker intercepts and decrypts ciphertext and then returns plaintext to the user.

Table 1: Example of QQMail features in the library.

Application	Function	Request method	Request URI	Request content type	Encryption field	Encryption algorithm
QQMail	sendMail	POST	set3.mail.qq.com/cgi-bin/compose_send?sid=JIq-6XB24WTmf0ke	Key-value format	Subject content	AES-256
QQMail	receiveMail	GET	set3.mail.qq.com/cgi-bin/readmail?folderid=1&folderkey=1&t=readmail&mailid=ZC0010-3nAORN1KF5ITb1kOXm	HTML format	Response data	AES-256

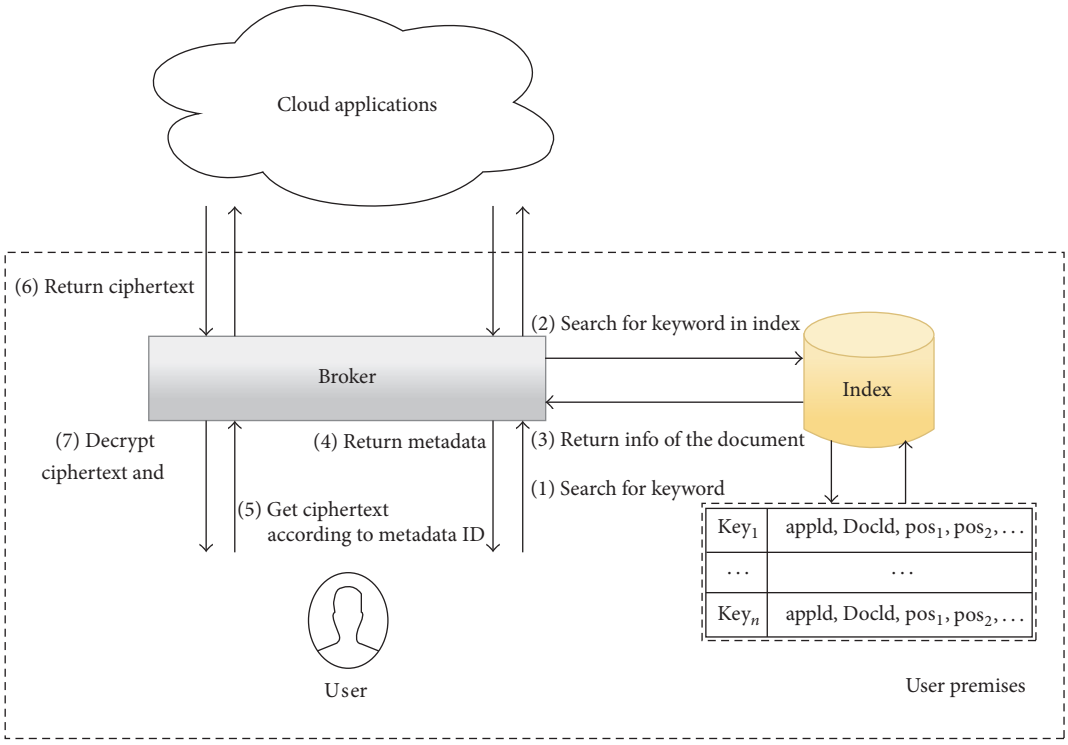


Figure 6: Architecture of ciphertext search in Enc-YUN.

3.4. Key and Metadata Management. Encrypted data sharing between users becomes difficult because the users do not share their own private key. Enc-YUN resolves this challenge by wrapped key. Each user has a private/public-key pair. The Key Manager stores the private key of the user, encrypted with the user's password. When the encrypter encrypts sensitive data, the Key Manager generates a random file-key which is used to encrypt the data. And Enc-YUN creates a wrapped key: an encryption of file-keys under the public key of users. If Alice wants to share a sensitive document with Bob, the Enc-YUN which needs to be authorized to obtain the private key of Alice unlocks the wrapped key and creates another wrapped key with Bob's public key. Thus, Bob will get plaintext of the document shared by Alice by unlocking the wrapped key with his private key.

The metadata linking the application functions and encrypted data is critical when the user in Enc-YUN authorizes another user outside of Enc-YUN to view encrypted mails and files. The metadata of a mail mainly includes components like mail id, sender id, recipient id, attachment id, ciphertext id, and so on.

4. Evaluation

In this section, we discuss the performance overhead of Enc-YUN. We conducted the test on an Intel 2.5 GHz × 2 with 2 GB of RAM. And we made some comparison tests between cloud storage and mail application under the circumstances of having Enc-YUN and not having Enc-YUN. The test could estimate the performance of the Enc-YUN by time-consuming brought from Enc-YUN.

Figure 7 shows that sending and receiving mails in Enc-YUN have more network overhead than in the normal network. Because the broker which is proxy would cost time to establish connections between users and cloud services, the time in which Enc-YUN recognizes protocols and analyzes semantic data from users had a small proportion in the entire

Table 2: Time cost in adding and viewing info in http://www.youshang.com/. "In" is the number of inputs which require encryption on the page. "Out" is the number of outputs which require decryption on the page.

Web page	In	Cost time of encrypter and parser (μs)	Cost time of adding info (μs)	Out	Cost time of encrypter and parser (μs)	Cost time of viewing info (μs)
Customer management	5	24407.8	247292 (9.87%)	10	140146.6	32576 (430%)
Supplier management	5	25025.8	57748 (43.34%)	20	222413.8	32536 (683%)
Good management	10	17043.2	75690 (22.52%)	10	112071.2	38380 (292%)
Warehouse management	1	9545.8	26490 (36.04%)	5	56961.4	25060 (227%)

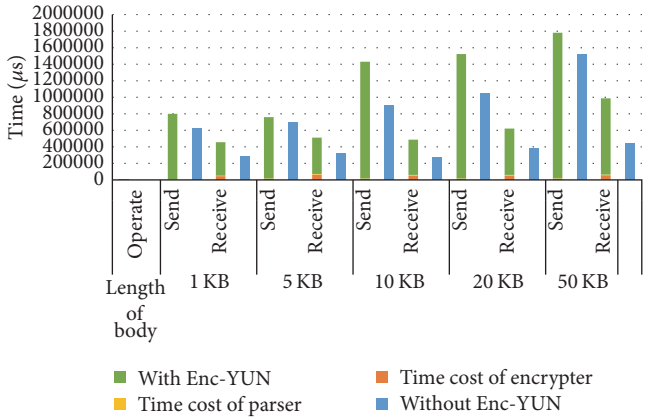


Figure 7: Time cost of Enc-YUN to send mails and receive mails in QQMail.

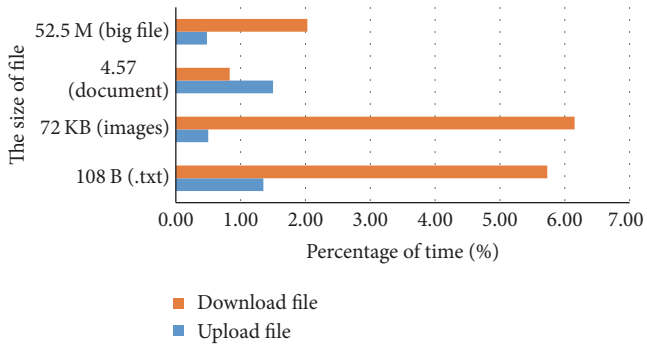


Figure 8: Percentage of time cost in encrypter of Enc-YUN to upload files and download files in Baidu Netdisk.

request completion time. This indicates that the algorithm of protocol recognition and semantic analysis in Enc-YUN is efficient. However, there are other time-consuming events besides the parser and encrypter. The time may be consumed by other modules of the Enc-YUN. We believe that the time cost will be less if every module of Enc-YUN has a good performance. And Figure 8 presents that the time percentage of uploading small files or documents is close to normal time of uploading files outside Enc-YUN. But the larger the file, the more the time it will consume. It indicates that the performance of Enc-YUN depends on the performance of the broker. Table 2 lists the number of input fields which require encryption for each page and the estimated time cost of encrypter and parser increase in microseconds. The page is loaded by browser in several hundred milliseconds. The performance overhead induced by the Enc-YUN is minor and acceptable according to user's perceptual experience in the millisecond level.

We also tested the Enc-YUN on a wide variety of popular SaaS applications such as Salesforce, Gmail, and Google Drive. And the application of message is the only type of application in which Enc-YUN cannot encrypt data because of the fact that the protocol of application is encrypted by providers. While encrypting data always impacts some application functionalities, we find that, for a broad range of applications, encrypting data still retains prominent functionality. As it can be seen from Table 3, Enc-YUN supports more semantic-rich functionalities because Enc-YUN searches in local index and retains the functionalities of normal information retrieval. All of the SSE schemes focus on text-formed data regardless of complex data structures in reality. However, Enc-YUN supports XML, JSON, relational database model, and so on. Thus the real time of search in Enc-YUN is also better than KPR [16] and KP [17] scheme.

5. Conclusion and Future Work

We presented Enc-YUN, a system for transparently encrypting confidential data and supporting selective and searchable data encryption. Without any modification of the cloud and applications, ciphertext search could take place in the Enc-YUN. Moreover, Enc-YUN has ability to support more and more applications with protocol recognition and semantic analysis.

Enc-YUN's contribution lies in providing a new perspective to achieving practical ciphertext search. And Enc-YUN's secure infrastructure and usable interface design provide a basis for implementing wide variety of encryption schemes. In the long run, we will try to improve the performance of transforming large files and we aim at supporting more automated protocol inspection and intelligent protocol analysis.

Competing Interests

The authors declare that they have no competing interests.

Table 3: Comparison between SE schemes and Enc-YUN. PEKS scheme: public-key encryption with keyword search scheme; SEKS scheme: symmetric-key encryption with keyword search scheme. m and n are the maximum # of keywords and files, k is the # of unique keywords included in an updated file (added or deleted), d is the # of incremental keywords in an updated file, p is the # of parallel processors, and t is the network latency introduced due to the interactions.

Category	Schemes	Query support						Update time
		Single keyword	Multiple keywords	Fuzzy search	Ranked search	Support complex data structures	Dynamic search	
SEKS schemes	SWP [18]	✓	✘	✘	✘	✘	✘	—
PEKS schemes	BCO+ [19]	✓	✘	✘	✘	✘	✘	—
	SLN+ [20]	✓	✘	✓	✘	✘	✘	—
Index-based SEKS schemes	Goh [21]	✓	✘	✘	✘	✘	✘	—
	GSW [22]	✓	✓	✘	✘	✘	✘	—
	KIK [23]	✓	✘	✓	✓	✘	✘	—
	XWS [24]	✓	✓	✘	✓	✘	✘	—
	KPR [16]	✓	✘	✘	✘	✘	✓	$O(k)$
	KP [17]	✓	✘	✘	✘	✘	✓	$O((m/p)\log n) + t$
Index-based PEKS schemes	RT [25]	✓	✓	✘	✘	✘	✘	—
	BW [26]	✓	✓	✘	✘	✘	✘	—
Enc-YUN	Enc-YUN	✓	✓	✓	✓	✓	✓	$O(d)$

Acknowledgments

This paper is supported by the National High Technology Research and Development Program of China (863 Program) under Grant no. 2015AA016001, the National Natural Science Foundation of China under Grant no. 61370068, Innovation Projects in Shandong Province under Grant no. 2014ZZCX03411, and Production-Study-Research Cooperation Project in Guangdong Province under Grant no. 2016B090921001.

References

[1] Cisco Visual Networking Index (VNI) Global Mobile Data Traffic Forecast for 2014 to 2019.

[2] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," *Journal of Computer Security*, vol. 19, no. 3, pp. 367–397, 2011.

[3] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[4] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 36–49, Berkeley, Calif, USA, May 2000.

[5] Y. C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security*, vol. 3531 of *Lecture Notes in Computer Science*, pp. 442–455, Springer, Berlin, Germany, 2005.

[6] D. Boneh, G. D. Crescenzo, and R. Ostrovsky, "Public key encryption with keyword search," in *Proceedings of the EUROCRYP '04*, pp. 506–522, Interlaken, Switzerland, May 2004.

[7] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology—EUROCRYPT 2004*, pp. 506–522, Springer, Berlin, Germany, 2004.

[8] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 262–267, 2011.

[9] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*, pp. 563–574, Paris, France, 2004.

[10] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: improved security analysis and alternative solutions," in *Advances in Cryptology—CRYPTO 2011*, vol. 6841 of *Lecture Notes in Computer Science*, pp. 578–595, Springer, Berlin, Germany, 2011.

[11] W. He, D. Akhawe, S. Jain, E. Shi, and D. Song, "ShadowCrypt: encrypted web applications for everyone," in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS '14)*, pp. 1028–1039, ACM, Scottsdale, Ariz, USA, November 2014.

[12] B. Lau, S. Chung, C. Song et al., "Mimesis aegis: a mimicry privacy shield—a system's approach to data privacy on public cloud," in *Proceedings of the 23rd USENIX Security Symposium (USENIX Security '14)*, pp. 33–48, August 2014.

[13] Virtru, https://www.virtru.com/.

[14] R. A. Popa, E. Stark, S. Valdez et al., "Building web applications on top of encrypted data using Mylar," in *Proceedings of the ACM 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pp. 85–100, Seattle, Wash, USA, April 2014.

[15] M. Abdalla, M. Bellare, D. Catalano et al., "Searchable encryption revisited: consistency properties, relation to anonymous IBE, and extensions," *Journal of Cryptology*, vol. 21, no. 3, pp. 350–391, 2008.

[16] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '12)*, pp. 965–976, New York, NY, USA, 2012.

[17] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography and Data Security*, vol. 7859 of *Lecture Notes in Computer Science*, pp. 258–274, Springer, Berlin, Germany, 2013.

[18] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 44–55, May 2000.

[19] D. Boneh, G. Di Crescenzo, R. Ostrovsky et al., "Public key encryption with keyword search," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 506–522, Springer, Berlin, Germany, 2004.

[20] S. Sedghi, P. van Liesdonk, S. Nikova, H. Pieter, and W. Jonker, "Searching keywords with wildcards on encrypted data," in *Proceedings of the 7th Conference on Security and Cryptography for Networks (SCN '10), Amalfi, Italy, September 2010*, vol. 6280 of *Lecture Notes in Computer Science*, pp. 138–153, Springer, 2010.

[21] E.-J. Goh, "Secure indexes," IACR Cryptology ePrint Archive, https://eprint.iacr.org/2003/216.

[22] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proceedings of the 2nd International Conference on Applied Cryptography and Network Security (ACNS '04)*, pp. 31–45, Springer, Yellow Mountain, China, 2004.

[23] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proceedings of the IEEE 28th International Conference on Data Engineering (ICDE '12)*, pp. 1156–1167, Washington, DC, USA, April 2012.

[24] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 340–352, 2016.

[25] E. K. Ryu and T. Takagi, "Efficient conjunctive keyword-searchable encryption," in *Proceedings of the IEEE 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW '07)*, vol. 1, pp. 409–414, Niagara Falls, Canada, May 2007.

[26] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of the 4th Theory of Cryptography Conference (TCC '07), Amsterdam, The Netherlands, February 2007*, vol. 4392 of *Lecture Notes in Computer Science*, pp. 535–554, Springer, 2007.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 3252148, 9 pages
http://dx.doi.org/10.1155/2016/3252148

Research Article

Distributed Parallel Endmember Extraction of Hyperspectral Data Based on Spark

Zebin Wu,[1,2] Jinping Gu,[1] Yonglong Li,[1] Fu Xiao,[2] Jin Sun,[1] and Zhihui Wei[1]

[1] *School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China*
[2] *Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks, Nanjing 210003, China*

Correspondence should be addressed to Zebin Wu; zebin.wu@gmail.com

Received 24 February 2016; Revised 6 May 2016; Accepted 22 May 2016

Academic Editor: Laurence T. Yang

Due to the increasing dimensionality and volume of remotely sensed hyperspectral data, the development of acceleration techniques for massive hyperspectral image analysis approaches is a very important challenge. Cloud computing offers many possibilities of distributed processing of hyperspectral datasets. This paper proposes a novel distributed parallel endmember extraction method based on iterative error analysis that utilizes cloud computing principles to efficiently process massive hyperspectral data. The proposed method takes advantage of technologies including MapReduce programming model, Hadoop Distributed File System (HDFS), and Apache Spark to realize distributed parallel implementation for hyperspectral endmember extraction, which significantly accelerates the computation of hyperspectral processing and provides high throughput access to large hyperspectral data. The experimental results, which are obtained by extracting endmembers of hyperspectral datasets on a cloud computing platform built on a cluster, demonstrate the effectiveness and computational efficiency of the proposed method.

1. Introduction

Hyperspectral remote sensing images are characterized by their large dimensionalities and volumes, with hundreds of nearly contiguous spectral channels. The hyperspectral image obtained from the earth's surface contains abundant information of space, radiation, and spectrum, which provides great help to the researchers for analyzing, processing, and monitoring the earth's surface information. However, due to the limitation of the sensor in spatial resolution and the diversity of the ground cover, the pixels of the image are generally mixed pixels. One of the most important techniques for hyperspectral data exploitation is endmember extraction [1], which characterizes mixed pixels as a combination of spectrally pure components (i.e., endmembers). Under the assumption of minimal secondary reflections and multiple scattering effects in data collection procedure, a number of techniques have been developed under the linear unmixing model in recent years [2], such as iterative error analysis (IEA) [3], independent component analysis (ICA) [4], dependent component analysis (DECA) [5], vertex component analysis (VCA) [6], simplex growing algorithm (SGA) [7], and minimum volume simplex analysis (MVSA) [8].

The abovementioned works have improved accuracy of hyperspectral endmember extraction enormously. However, most of them are very computationally intensive and therefore compromise their applicability in time-critical scenarios including military reconnaissance, environmental quality surveillance, monitoring of chemical contamination, wildfire tracking, and biological threat detection. As a result, in recent years, many techniques have been developed towards the improvement of these algorithms in high-performance computing architectures [9, 10]. For instance, low-weight integrated components such as field programmable gate arrays (FPGAs) [11], multicore central processing units (CPUs) [12, 13], and commodity graphics processing units (GPUs) [14, 15] have been successfully applied to accelerate computations. Nevertheless, with the development of hyperspectral imaging technology and the volume of the hyperspectral image growing, the traditional mechanism of allocating computational resources to a single machine is insufficient to meet the requirements of efficient hyperspectral processing. Accordingly, the fast endmember extraction of large hyperspectral dataset has been an important issue in the field of hyperspectral remote sensing. Fortunately, cloud computing has recently become more and more popular in the research

Input: Hyperspectral data $\mathbf{X}_{N\times L}$, the number of endmembers m.
(1) Initialization: Threshold of θ (spectral angle), the number R for averaging the vectors with the largest error, and endmember matrix $\mathbf{U}$.
(2) Calculate the mean vector $\mathbf{mean}_{1\times L}$ of the hyperspectral data $\mathbf{X}_{N\times L}$.
(3) Perform the constrained unmixing on $\mathbf{X}_{N\times L}$ using $\mathbf{mean}_{1\times L}$ as endmember matrix, and get the image of the errors (named "error image") remaining after the unmixing.
repeat
(4) Find R pixel vectors with the largest error in the error image, and extract the subset of the set of R vectors, consisting of all those vectors which fall within the angle θ of the maximum error vector.
(5) Average the vectors in the subset to decrease the effects of outliers and noise, denoted as $\mathbf{p}_i$, and update endmember matrix $\mathbf{U} = [\mathbf{U}; \mathbf{p}_1]$.
until m endmembers are extracted.

Algorithm 1: Endmember extraction based on IEA.

and commercial fields due to its homogeneous operating environment and full control over dedicated resources (e.g., networks, servers, storage, applications, and services) [16, 17]. Cloud computing can be considered as the improved processing for distributed processing, parallel processing, and grid computing [18]. However, to the best of our knowledge, despite the potential of large-scale distributed parallel computing in cloud computing and the demands of massive data processing in hyperspectral imaging, there are few cloud computing implementations of this category of algorithms in the literatures. In order to efficiently extract endmembers from massive hyperspectral data, a novel distributed parallel endmember extraction method based on iterative error analysis (IEA_DP) is proposed by utilizing cloud computing principles to efficiently process massive hyperspectral data. In particular, the storage of hyperspectral data is well organized to reduce the correlation among data partitions as well as to avoid data skew. The processing logic of IEA algorithm is optimized by reducing the intermediate data generated by each execution node and avoiding transitional large data. The newly developed method is implemented and evaluated on Spark and MapReduce model. Its efficiency is evaluated in terms of accuracy and parallel execution performance through the comparison with a serial IEA implementation on a single CPU.

2. Endmember Extraction Based on IEA

Let $\mathbf{X}_{N\times L} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T \in \mathbf{R}^{N\times L}$ denote a hyperspectral image with N pixels, where $\mathbf{x}_i \in \mathbf{R}^L$ is an L-dimensional hyperspectral pixel observation. The linear mixture model identifies a collection of spectrally pure constituent spectra (endmembers) and expresses the measured spectrum of a mixed pixel as a linear combination of the endmembers, weighted by fractional abundances that indicate the proportion of each endmember contained by the pixel [1]. This procedure can be described in mathematical terms as follows:

$$\mathbf{x}_i = \mathbf{U}\mathbf{f}_i + \mathbf{n}, \tag{1}$$

where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m]$ denotes an L-by-m mixing matrix in which the endmembers correspond to the columns. This matrix is in general of full column rank. Here, m denotes the number of endmembers, $\mathbf{f}_i = [f_{i1}, f_{i2}, \ldots, f_{im}]^T$ denotes an m-by-1 vector containing the respective fractional abundances of the endmembers, f_{ik} is the abundance fraction of the kth endmember, with $k = 1, 2, \ldots, m$, and the notation $(\cdot)^T$ stands for vector transpose operation; $\mathbf{n}$ denotes an additive L-by-1 noise vector representing the errors that affect the measurement of the pixel at each spectral band. Endmember extraction of hyperspectral data aims at obtaining a good estimation of the mixing matrix $\mathbf{U}$. Several methods have been used to perform endmember extraction, including geometrical, statistical, and sparse regression-based approaches [1]. Among these methods, the IEA algorithm is one of the most successful algorithms of the first category and therefore has been widely used.

Assuming the existence of relatively pure pixels, the IEA algorithm performs a series of linear constrained unmixing [19] and chooses endmembers by minimizing the remaining error in the unmixed image [3]. This procedure is executed directly on the spectral data, without the requirement of transformation into Principal Components (PCs) or any other elimination of redundancy. A step-by-step description of the IEA algorithm is given as shown in Algorithm 1.

3. Processing Framework Based on Cloud Computing

In general, cloud computing uses MapReduce programming model, which is essentially a coarse granularity parallel programming model. MapReduce model can automatically parallelize the large-scale computing tasks. More importantly, the implementation details are transparent to the users. Users define the computation of map and reduce functions, and the underlying operating system automatically performs the parallel computation across large-scale clusters of machines and makes efficient use of network and disks by scheduling intermachine communication [20]. Hadoop, a cloud computing framework that uses MapReduce model, is well known for its fault tolerance and scalability [21]. However, Hadoop

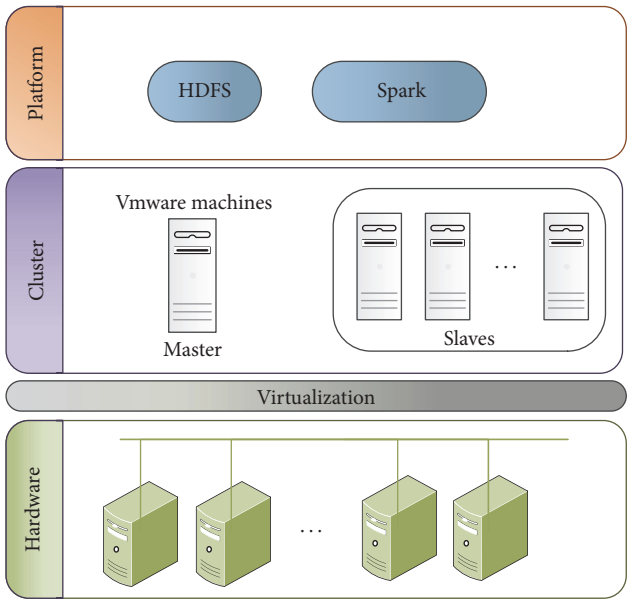


Figure 1: The processing framework based on cloud computing.

solutions rely on writing and reading data from HDFS and therefore are of slow speed.

Fortunately, Apache Spark, a novel high-performance framework capable of tackling massive data processing workloads while coping with larger and larger scales, has been proposed in [22]. This framework enables streaming and interactive queries and demonstrates its scalability, fault tolerance, and the ability of handling batch processing. Apache Spark is a cluster-computing platform, which is open source, Hadoop-compatible, fast, and expressive. In terms of data storage, Spark abstracts out of the distributed memory storage structure by Resilient Distributed Dataset (RDD) [23]. The RDD can control the data in the partition of different nodes and is compatible with HDFS. A large amount of existing data in HDFS can be loaded into RDD for being processed as a data source. Spark runs on top of existing HDFS infrastructures to provide enhanced and additional functionality. Moreover, Spark is based on memory computing, which holds intermediate results in memory rather than writing them to HDFS.

To be specific, the processing framework for distributed parallel endmember extraction of hyperspectral data based on cloud computing can be summarized graphically as shown in Figure 1.

4. Distributed Parallel Optimization for IEA Based on Spark

Spark is an extensible platform for data analysis that integrates the calculation of primitive memory. Therefore, Spark achieves better performance compared with Hadoop cluster storage methods. With the development of remote sensing technology, the data quantity of hyperspectral remote sensing data is increasing. Even a single pixel may contain hundreds of spectral information types, leading to more difficulty in the calculation of hyperspectral data processing. On the other hand, accelerating the processing of large amounts of data in hyperspectral imagery is of great importance. In this work, we present a distributed parallel implementation of IEA algorithm (IEA_DP) for endmember extraction of massive hyperspectral image based on Spark.

It can be observed that the most time-consuming parts of Algorithm 1 are the procedure of constrained unmixing, the calculation of error image, and the selection of the vectors with the largest error. Therefore, we concentrate on the parallel optimization of these parts. In what follows, we describe the distributed implementation of the different phases of Algorithm 1 and describe the architecture-level optimizations performed during the development of the parallel implementation.

Storage is a critical issue of our distributed parallel implementation. In hyperspectral remote sensing, with increasingly growing data volumes, it is important to efficiently store and utilize potentially unlimited amounts of hyperspectral datasets. HDFS represents a perfect choice for the reliability and elasticity of the task of storing very large files on different resources on large clusters. As a result, we store the original hyperspectral datasets on HDFS, taking advantage of its capabilities for distributed storage, fault tolerance, and flexibility in a transparent way.

A class (named `HSIInputFormat`) is defined to read original hyperspectral datasets from HDFS to `NewHadoopRDD` instances `ByteRDD`. In HDFS, the original hyperspectral image is divided into many spatial-domain partitions [16]. In order to reduce the I/O (input/output) overhead to the most extent, we read every data partition on HDFS as a `key-value` pair, in which the key (named `Offset`) is the offset of this partition in the original dataset and the value (named `Pixels`) is the hyperspectral data partition of byte type. Subsequently, `ByteRDD` is mapped onto a `MapPartitionsRDD` (which consists of formatted pixel vectors data, denoted as `DataRDD`). Finally, we cache `DataRDD` in RAM for fast access. The flowchart of the procedure for reading hyperspectral datasets is graphically illustrated in Figure 2.

Firstly, the `DataRDD` is mapped onto partitions $\mathbf{Pixel}_{p\times L}$, which are accumulated to $\mathbf{Sum}_{1\times L}$ by column. The `reduce` operation is then performed to aggregate theses $\mathbf{Sum}_{1\times L}$, and the mean vector $\mathbf{mean}_{1\times L}$ of the hyperspectral data $\mathbf{X}_{N\times L}$ is calculated on driver.

Secondly, the constrained unmixing is performed on $\mathbf{X}_{N\times L}$ using $\mathbf{mean}_{1\times L}$, as follows:

$$\mathbf{abu}_{1\times L}^{T} = \underbrace{\left(\mathbf{I} - \frac{\left(\mathbf{E}\mathbf{E}^{T}\right)^{-1}\mathbf{a}\mathbf{a}^{T}}{\mathbf{a}^{T}\left(\mathbf{E}\mathbf{E}^{T}\right)^{-1}\mathbf{a}}\right)\left(\mathbf{E}\mathbf{E}^{T}\right)^{-1}}_{\mathbf{ta}}\mathbf{E}\mathbf{x}_{1\times L}^{T} + \underbrace{\frac{\left(\mathbf{E}\mathbf{E}^{T}\right)^{-1}\mathbf{a}}{\mathbf{a}^{T}\left(\mathbf{E}\mathbf{E}^{T}\right)^{-1}\mathbf{a}}}_{\mathbf{tb}}, \tag{2}$$

where $\mathbf{I}$ is an N-order identity matrix, $\mathbf{a}$ is an N-dimensional column vector with all 1 entries, $\mathbf{E} = \mathbf{mean}_{1\times L}$, and $\mathbf{x}$ is a pixel vector.

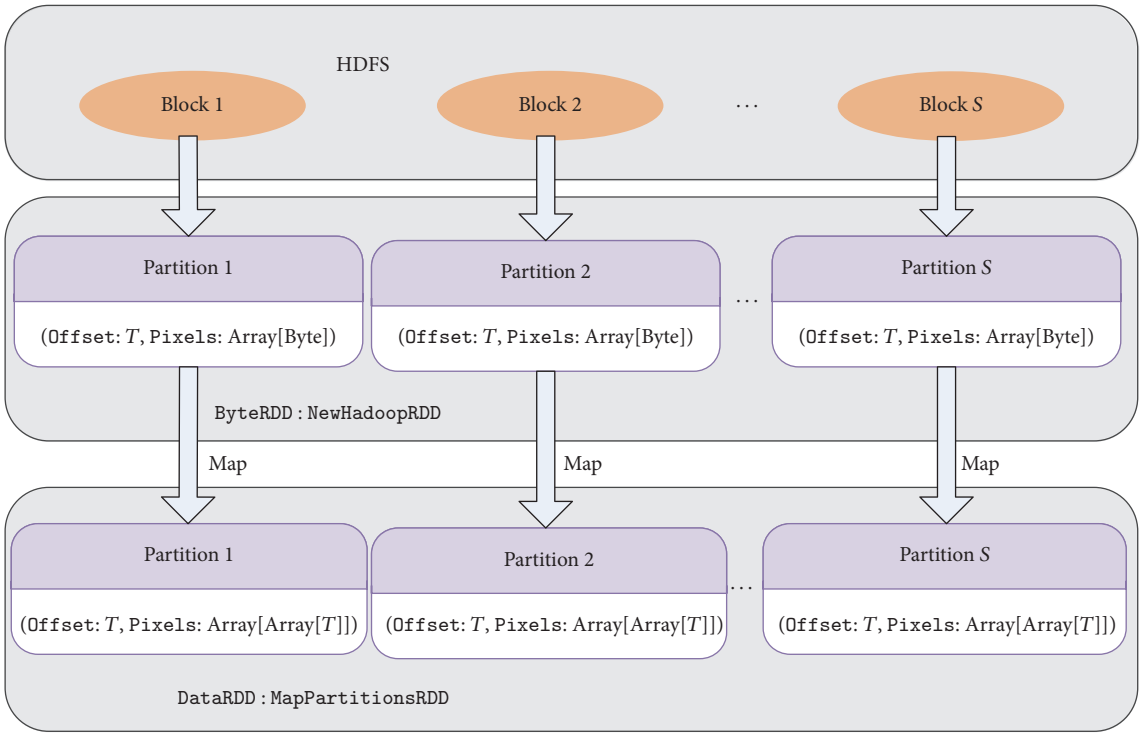


Figure 2: Graphical illustration of the procedure used for reading hyperspectral datasets.

When computing (2), **ta** and **tb** parts are calculated, respectively, on driver. After that, **ta**, **tb,** and **E** are broadcasted to all workers. To avoid frequent garbage collection of Java virtual machine, pixels in the partition are processed successively by `map` operation. Take the pixel vector $\mathbf{Pixel}_i$ for example; its abundance coefficients are estimated by (2) and stored in vector $\mathbf{abu}_{1\times N}$. Then, its reconstruction error can be computed. When all the pixels in the partition have been processed, the maximum error max, as well as the corresponding position $pos_{\max}$, is selected. A tuple ($pos_{\max}$ + offset, max, $\mathbf{p}_{\max}$) is obtained as the output of `map` operation, where offset is the key of the certain partition that denotes the position of its first pixel in the whole hyperspectral image. The tuple with maximum max is afterwards selected by the `reduce` operation and is returned to driver.

In the next procedure, **ta**, **tb**, **E**, **pos**, and R are transmitted to every worker as broadcast variables. The `map` operations are performed on `DataRDD` to find the R pixels with the largest errors, and the subset of the set of R vectors is extracted, which consists of all those vectors which fall within the spectral angle θ of the maximum error vector. After the `reduce` operation is performed to merge sort, the vectors in the subset is averaged as $\mathbf{p}_i$ to decrease the effects of outliers and noise, and endmember matrix is updated ($\mathbf{U} = [\mathbf{U}; \mathbf{p}_1]$) on driver side.

The algorithmic details of IEA_DP are graphically illustrated in Figure 3. The storage of hyperspectral data is well organized to reduce the correlation among partitions as well as to avoid the data skew. Moreover, the logical procedure of IEA algorithm is optimized by reducing the intermediate data generated by every execution node and avoiding transitional large data. Accordingly, the computation of the IEA algorithm can be greatly accelerated.

5. Experimental Evaluation

To evaluate the proposed distributed parallel implementation of IEA algorithm, experiments were performed on a Spark equipped cluster with 1 master node and 8 slave nodes. Master is also the NameNode of HDFS and slaves are the DataNodes of HDFS. Master is a virtual machine created on the host with an Intel Xeon E5630 CPUs at 2.53 GHz with 8 cores by the VMware vSphere. The slave nodes are implemented by virtual machines created based on the virtualization of a 4-blade IBM Blade Center HX5 equipped with 2 Intel Xeon E7-4807 CPUs at 1.86 GHz and connected to a 12 TB disk array by SAS bus. Each slave is configured with 6 CPU cores. Master and slaves are all installed with Ubuntu 12.04, Hadoop 1.2.1, Spark 1.4.1, and Java 1.6.45. In addition, all nodes are connected by a gigabit switch. Figure 4 illustrates the architecture of the experimental platform.

The hyperspectral dataset used in our experiments is a subset of the well-known Airborne Visible Infrared Imaging Spectrometer (AVIRIS) Cuprite (http://aviris.jpl.nasa.gov/data/free_data.html) image with 224 spectral bands, which was collected over the Cuprite mining site, Nevada, in 1995. This scene has been widely used to validate the performance of endmember extraction algorithms. Some bands 1–3, 107–114, 159–169, and 221–224 have been removed prior to

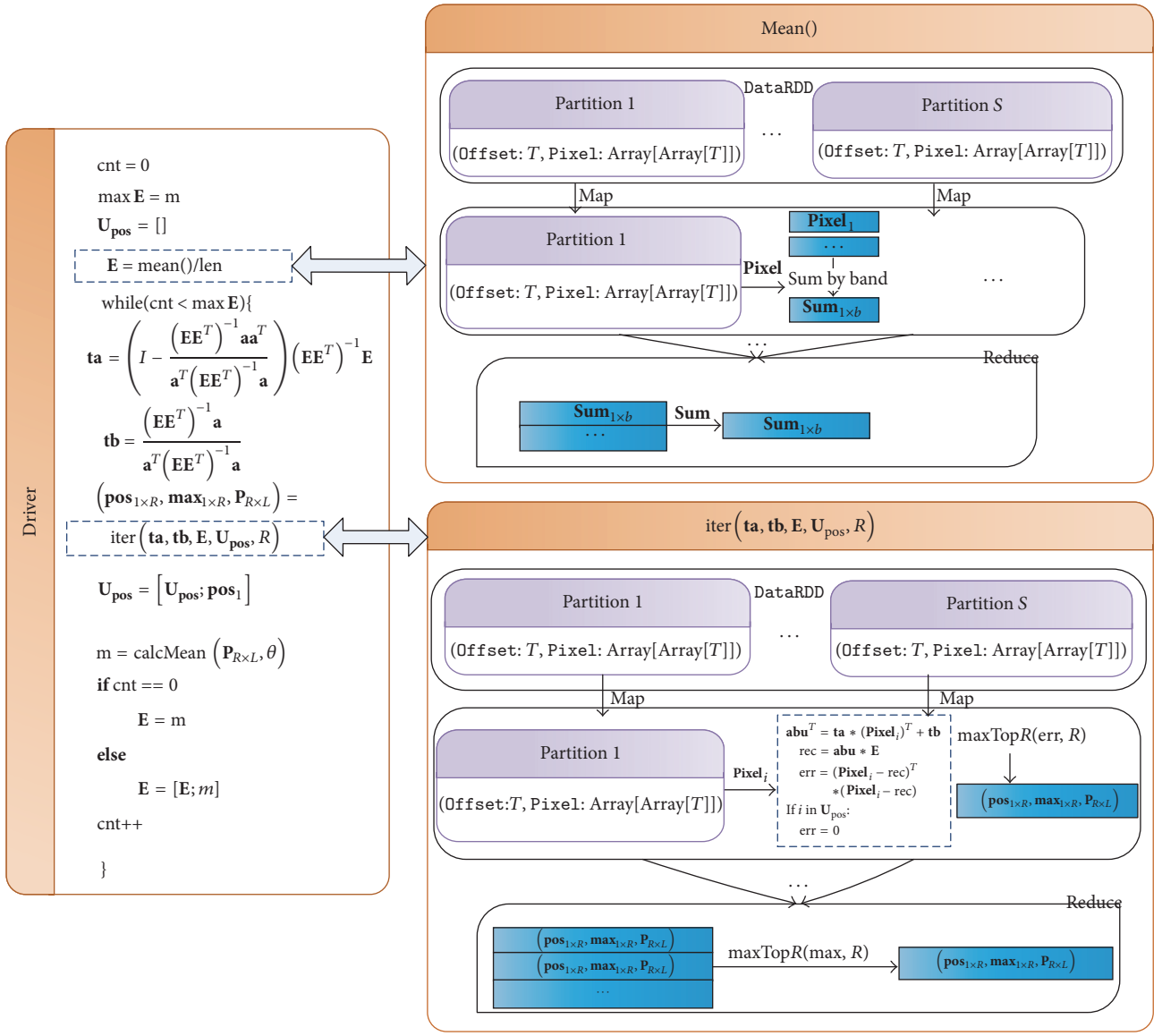


Figure 3: Design of IEA_DP algorithm based on Spark.

the analysis due to water absorption bands and bands with low SNR. The selected dataset consists of 350 × 350 pixels (as shown in Figure 5), 192 bands, and a total size of about 44.86 MB. In order to evaluate the algorithm's performance on data of different sizes, we use the Mosaicking function of ENVI software to generate 3 datasets with different sizes (including Dataset 1: 8400 × 350 with the size of about 1.05 GB, Dataset 2: 16800 × 350 with the size of about 2.10 GB, and Dataset 3: 33600 × 350 with the size of about 4.21 GB) by mosaicking the original 44.86 MB dataset. Both computational performance and unmixing accuracy have been taken into consideration for evaluation.

At the beginning, we evaluate the accuracy of the considered endmember extraction algorithm on the AVIRIS Cuprite image, taking advantage of the availability of detailed laboratory measurements of endmembers contained in the scene (http://speclab.cr.usgs.gov/maps.html). According to the survey results by the US Geological Survey (USGS), the Cuprite mining site mainly contains five categories of minerals, that is, Alunite, Buddingtonite, Calcite, Kaolinite, and Muscovite. Their reference ground signatures are available in the form of a USGS library (http://speclab.cr.usgs.gov/spectral-lib.html). The most similar endmember signatures extracted by the IEA algorithm are selected and compared with the available 5 reference USGS spectral signatures in terms of spectral angle distance (SAD), measured in radians. According to Table 1, it can be concluded that the serial and distributed parallel versions of IEA obtain identical results, while the extracted endmembers are very similar, spectrally, to the reference USGS spectra.

The most important aspect in this work is to what extent the distributed parallel implementation improves the endmember extraction procedure in terms of computational performance. Before reporting our performance evaluation,

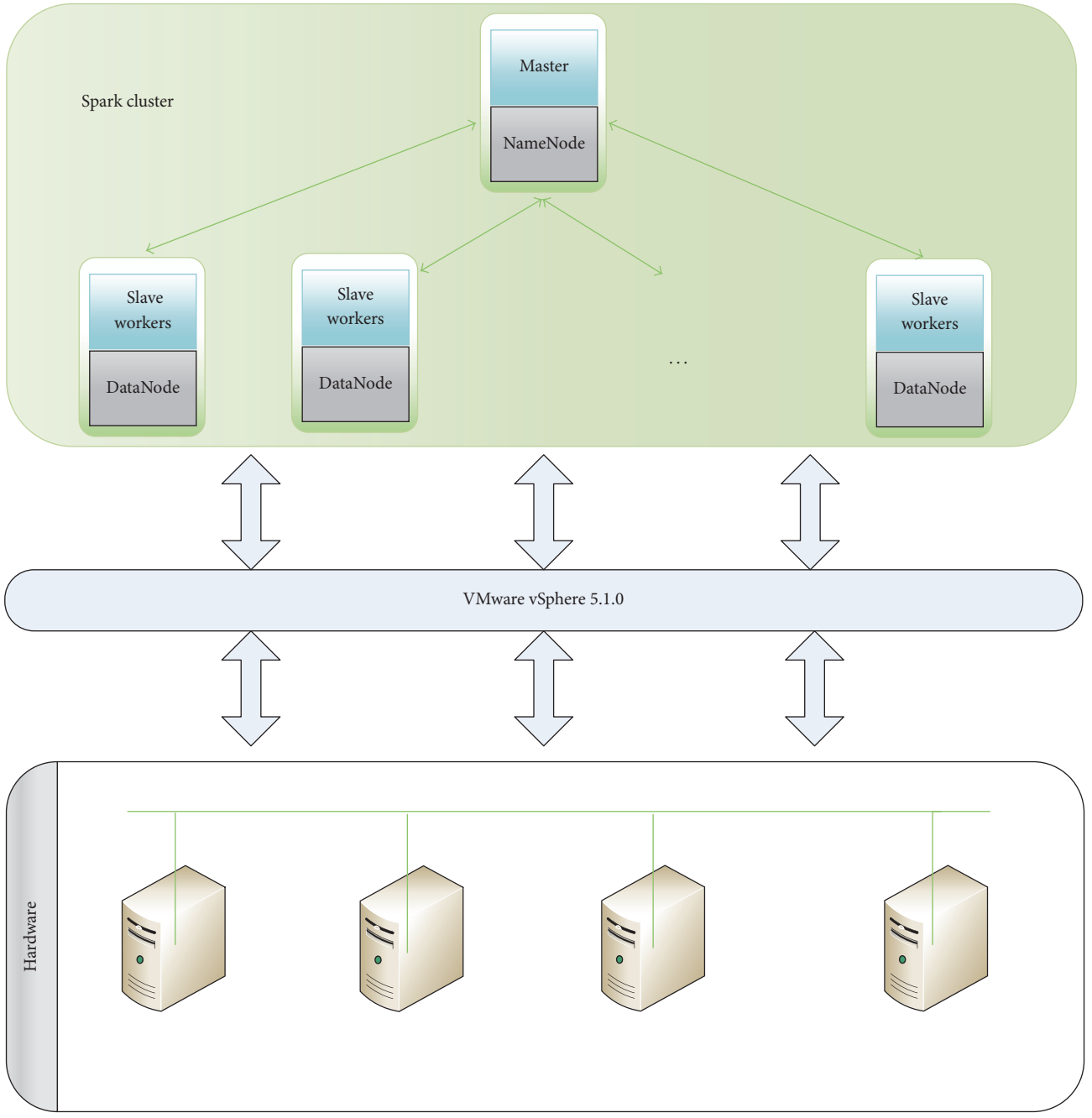


Figure 4: The architecture of the experimental platform.

it is worth emphasizing that our parallel implementation provides identical results as the serial version in terms of endmember spectra and fractional abundances. The key difference between the serial and parallel versions is the required runtime for completing the calculation. In subsequent part, we report the computational performance of the two versions executed on the datasets with different sizes, that is, Dataset 1, Dataset 2, and Dataset 3.

The execution time and speedups spent in processing Dataset 1 on the considered cloud computing architecture are listed in Table 2. In the first column of Table 2, "Parallel (x)" denotes the parallel version executed on a distributed platform consisting of 1 master node and x slave nodes (where $x = 1, 2, 4, 8$). As can be seen from Table 2, the execution time significantly decreases with regard to the increasing number of nodes. In terms of speedups, Figure 6 indicates approximately a linear growth with the number of nodes.

It is worth noting that the size of partition has great effects on the performance of the parallel versions. In other words, a good tuning of partition size helps improve the computational performance. In this paper, we empirically set the partition sizes (as Table 2 shows) to guarantee over 80%

Table 1: Spectral angle distance comparison between the endmembers extracted by IEA from the AVIRIS Cuprite scene and the reference USGS mineral spectral signatures.

Mineral	SAD (in radians)	
	Serial version	Parallel version
Alunite	0.0645	0.0645
Buddingtonite	0.0717	0.0717
Calcite	0.0898	0.0898
Kaolinite	0.0899	0.0899
Muscovite	0.0736	0.0736

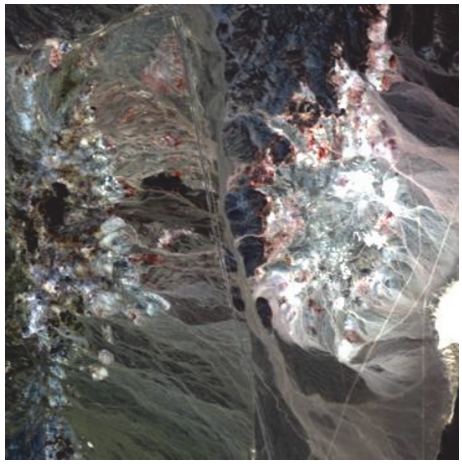


Figure 5: A 350 × 350-pixel subset of the AVIRIS Cuprite scene.

CPU utilization for every slave while running, thus leading to promising performance and speedup.

In a similar manner, experiments were performed on Dataset 2 and Dataset 3, using one master node and 8 slaves, to evaluate the efficiency of IEA_DP algorithm on larger datasets, as well as to compare IEA_DP algorithm with a Hadoop based IEA algorithm. It can be concluded from Figure 7 that the execution time of the proposed IEA_DP algorithm scales roughly linearly with the size of the dataset. Moreover, it is computationally efficient for large datasets (e.g., only 8.1%, and 6.2% of the time is consumed by initialization for the execution on Dataset 2 and Dataset 3, resp.). This conclusion is important, as it indicates the better scalability of the proposed IEA_DP as the data size increases. Moreover, Figure 8 demonstrates that the IEA_DP algorithm executed on the Spark platform processes the hyperspectral data much faster than on the Hadoop platform.

To summarize, the proposed IEA_DP performs and scales well under massive hyperspectral data. In particular, the availability of additional computing resources leads to more significant speedups. When using the cloud platform consisting of 1 master node and 8 slave nodes, the IEA_DP algorithm achieves a speedup of 33x in the hyperspectral endmember extraction. An endmember extraction task involving about 1 GB hyperspectral dataset, which took more than half an hour to be completed by the serial version, can now be completed in around 1.1 minutes by using our distributed parallel algorithm. This achievement is very promising for

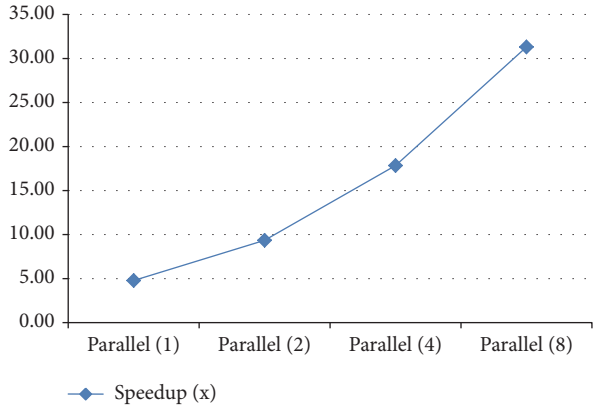


Figure 6: Speedup of the IEA_DP with Dataset 1.

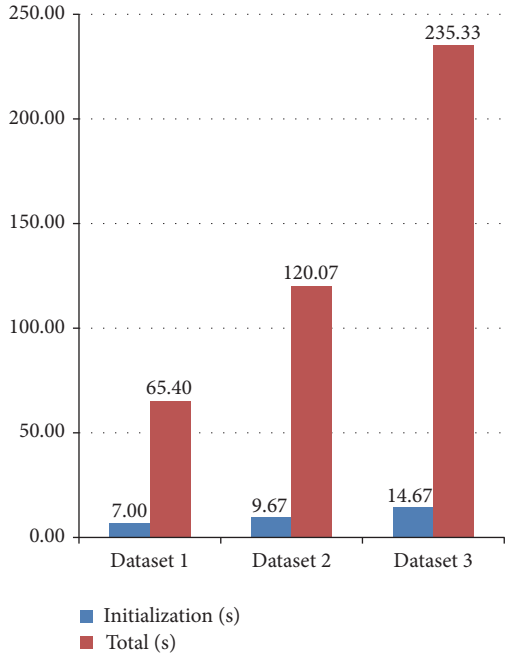


Figure 7: Execution time of the IEA_DP algorithm with different datasets.

a highly complex task, such as endmember extraction of a hyperspectral image with high volume and dimensionality.

6. Conclusions

The increased availability of high dimensional hyperspectral datasets is becoming an important challenge for hyperspectral image processing. This paper proposes a novel distributed parallel endmember extraction method based on iterative error analysis that utilizes advanced cloud computing technologies such as HDFS, Apache Spark, and the MapReduce model, to efficiently process massive hyperspectral data. Our experimental results indicate that the proposed method can be used to effectively process distributed collections of hyperspectral data of large scale. This contribution leads to the conclusion that hyperspectral image processing can greatly benefit from the efficient utilization of cloud computing

Table 2: Execution time of the serial and parallel versions of the IEA method with Dataset 1.

	Partition size (MB)	Initialization (s)	IEA (s)	Total (s)	Speedup (x)	Percentage of initialization
Serial	—	30.17	2017.68	2047.85	—	1.47%
Parallel (1)	269.17	12.33	415.80	428.13	4.78	2.88%
Parallel (2)	134.58	9.00	210.00	219.00	9.35	4.11%
Parallel (4)	67.29	8.67	106.13	114.80	17.84	7.55%
Parallel (8)	33.65	7.00	58.40	65.40	31.31	10.70%

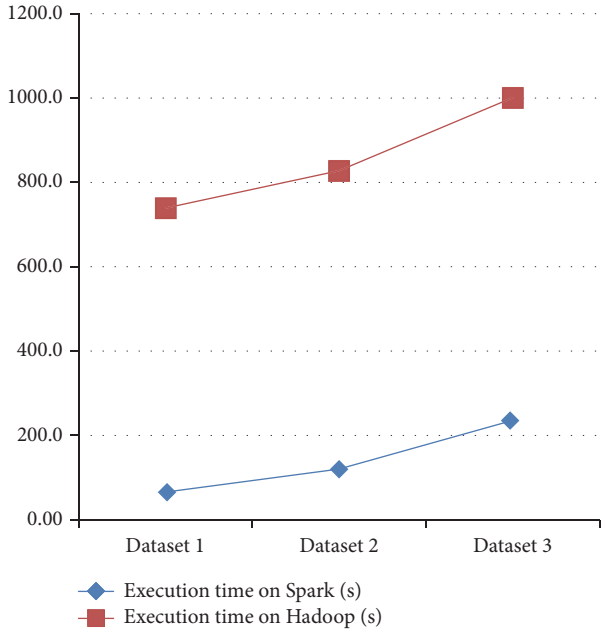


Figure 8: Execution time comparison between the Spark platform and the Hadoop platform.

architectures. Future work will focus on optimizing more complicated algorithms and applications for remotely sensed hyperspectral images.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

Financial support for this work, provided by the National Natural Science Foundation of China (Grant nos. 61471199, 91538108, and 11431015), the Research Fund of Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks under Grant no. WSNLBKF201507, the Jiangsu Province Six Top Talents Project of China under Grant no. WLW-011, is gratefully acknowledged.

References

[1] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon et al., "Hyperspectral unmixing overview: geometrical, statistical, and sparse regression-based approaches," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 2, pp. 354–379, 2012.

[2] Z. Wu, S. Ye, J. Liu, L. Sun, and Z. Wei, "Sparse non-negative matrix factorization on GPUs for hyperspectral unmixing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 8, pp. 3640–3649, 2014.

[3] R. A. Neville, K. Staenz, T. Szeredi et al., "Automatic endmember extraction from hyperspectral data for mineral exploration," in *Proceeding of 21st Canadian Symposium Remote Sensing*, pp. 21–24, Ottawa, Canada, June 1999.

[4] J. M. P. Nascimento and J. M. B. Dias, "Does independent component analysis play a role in unmixing hyperspectral data?" *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, no. 1, pp. 175–187, 2005.

[5] J. M. P. Nascimento and J. M. Bioucas-Dias, "Hyperspectral unmixing algorithm via dependent component analysis," in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS '07)*, pp. 4033–4036, Barcelona, Spain, June 2007.

[6] J. M. P. Nascimento and J. M. B. Dias, "Vertex component analysis: a fast algorithm to unmix hyperspectral data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, no. 4, pp. 898–910, 2005.

[7] C.-I. Chang, C.-C. Wu, W.-M. Liu, and Y.-C. Ouyang, "A new growing method for simplex-based endmember extraction algorithm," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 10, pp. 2804–2819, 2006.

[8] J. Li and J. B. Dias, "Minimum volume simplex analysis: a fast algorithm to unmixhyperspectral data," in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, pp. 250–253, Boston, Mass, USA, 2008.

[9] A. Plaza, Q. Du, Y.-L. Chang, and R. L. King, "High performance computing for hyperspectral remote sensing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 528–544, 2011.

[10] C. A. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: a review," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 508–527, 2011.

[11] C. González, D. Mozos, J. Resano, and A. Plaza, "FPGA implementation of the N-FINDR algorithm for remotely sensed hyperspectral image analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 2, pp. 374–388, 2012.

[12] A. Remón, S. Sánchez, A. Paz, E. S. Quintana-Ortí, and A. Plaza, "Real-time endmember extraction on multi-core processors," *IEEE Geoscience and Remote Sensing Letters*, vol. 8, no. 5, pp. 924–928, 2011.

[13] S. Bernabe, S. Sanchez, A. Plaza, S. Lopez, J. A. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: a comparison," *IEEE Journal of Selected*

Topics in Applied Earth Observations and Remote Sensing, vol. 6, no. 3, pp. 1386–1398, 2013.

[14] A. Barberis, G. Danese, F. Leporati, A. Plaza, and E. Torti, "Real-time implementation of the vertex component analysis algorithm on GPUs," *IEEE Geoscience and Remote Sensing Letters*, vol. 10, no. 2, pp. 251–255, 2013.

[15] J. M. P. Nascimento, J. M. Bioucas-Dias, J. M. Rodriguez Alves, V. Silva, and A. Plaza, "Parallel hyperspectral unmixing on GPUs," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 3, pp. 666–670, 2014.

[16] Z. Wu, Y. Li, A. Plaza, J. Li, F. Xiao, and Z. Wei, "Parallel and distributed dimensionality reduction of hyperspectral data on cloud computing architectures," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 6, pp. 2270–2278, 2016.

[17] Z. Chen, N. Chen, C. Yang, and L. Di, "Cloud computing enabled web processing service for earth observation data processing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 6, pp. 1637–1649, 2012.

[18] K. Stanoevska-Slabeva, T. Wozniak, and S. Ristol, *Grid and Cloud Computing: A Business Perspective on Technology and Applications*, Springer, Heidelberg, Germany, 2010.

[19] D. C. Heinz and C.-I. Chang, "Fully constrained least squares linear spectral mixture analysis method for material quantification in hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 3, pp. 529–545, 2001.

[20] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[21] D. Borthakur, "HDFS Architecture Guide," 2008, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf.

[22] M. Zaharia, M. Chowdhury, T. Das et al., "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI '12)*, p. 2, USENIX Association, San Jose, Calif, USA, April 2012.

[23] M. Zaharia, "An architecture for fast and general data processing on large clusters," Tech. Rep. UCB/EECS-2014-12, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2014.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 4386362, 15 pages
http://dx.doi.org/10.1155/2016/4386362

Research Article

A Two-Tier Energy-Aware Resource Management for Virtualized Cloud Computing System

Wei Huang,[1] Zhen Wang,[2] Mianxiong Dong,[3] and Zhuzhong Qian[2]

[1]*School of Computer Engineering, Nanjing Institute of Technology, Nanjing 211167, China*
[2]*State Key Lab. for Novel Software Technology, Nanjing University, Nanjing 210023, China*
[3]*Department of Information and Electronic Engineering, Muroran Institute of Technology, Muroran 050-8585, Japan*

Correspondence should be addressed to Zhuzhong Qian; qzz@nju.edu.cn

Received 22 February 2016; Accepted 1 September 2016

Academic Editor: Tomàs Margalef

The economic costs caused by electric power take the most significant part in total cost of data center; thus energy conservation is an important issue in cloud computing system. One well-known technique to reduce the energy consumption is the consolidation of Virtual Machines (VMs). However, it may lose some performance points on energy saving and the Quality of Service (QoS) for dynamic workloads. Fortunately, Dynamic Frequency and Voltage Scaling (DVFS) is an efficient technique to save energy in dynamic environment. In this paper, combined with the DVFS technology, we propose a cooperative two-tier energy-aware management method including local DVFS control and global VM deployment. The DVFS controller adjusts the frequencies of homogenous processors in each server at run-time based on the practical energy prediction. On the other hand, Global Scheduler assigns VMs onto the designate servers based on the cooperation with the local DVFS controller. The final evaluation results demonstrate the effectiveness of our two-tier method in energy saving.

1. Introduction

Cloud computing provides elastic computing resources on a pay-as-you-go basis for most conceivable forms of applications but it also causes huge amounts of electric energy consumption. Almost 0.5% of world's total power usage is consumed by the servers in data centers [1]. Among them, processors (CPUs) account for the most significant part of power and have the most dynamical power that can be adjusted, while other components can only be completely or partially turned off [2]. Owing to these reasons, reducing energy consumption of processors using the dynamic nature of CPUs' power has become a hot research topic in cloud computing system.

To service more users for more income, service providers prefer to share cluster resources among users. In cloud environments, the virtualization technique is widely adopted to allow users to share the physical resources. Making the working servers for Virtual Machines (VMs) as less as possible and letting the idle servers be in a low-power mode will improve the utilization of resources and reduce energy consumption, which is known as VM consolidation. In each server, by applying Dynamic Voltage and Frequency Scaling (DVFS), which enables dynamic adjustment of execution frequency on demand, more energy can be saved. The dynamic power consumption of CPU is proportional to the frequency and to the square of voltage. Scaling down the execution frequency will reduce the power while it may also reduce the performance and increase the execution time, which may instead cause more energy consumption (energy is equal to the line integral of power P to time t, $E = \int_0^t P\,dt$). On the other hand, real-time tasks in the cloud computing system usually have requirements on execution speed; the extension of execution time may violate QoS requirements. Thus, it is nontrivial to reduce energy consumption by scaling the execution frequencies of tasks [3].

VM consideration could improve the resource indeed and many previous works [4–6] achieve significant result on energy saving in virtualized cloud system. However, most of them do not take the advantage of DVFS strategy. Some others only apply the DVFS after allocation while not considering

the influence of DVFS before allocation. However, if taking impacts of DVFS technique on energy into consideration before allocation, much more energy can be reduced. But it is not trivial to minimize the total energy consumption by VM allocation algorithm and DVFS strategies in this way. Several problems need to be solved: (1) how to define Quality of Service (QoS) requirements; (2) which servers can load arrival VMs with requirements; (3) which server brings minimum energy consumption by DVFS for arrival VMs and ensures QoS requirements.

In this paper, we propose a cooperative two-tier energy-aware management by taking the DVFS into consideration, which offload real-time tasks to VMs on clusters and scaling frequencies. On the local tier, we propose a novel way to find the best combinations of frequencies of different CPUs that consume the least energy based on the practical energy prediction. We take both the frequency-power and utilization-power relationship into consideration when forecasting energy consumption. On the global tier, by cooperating with local DVFS controller, the Global Scheduler assigns a VM to its favourite processor in a cluster that brings minimum energy change. The frequency to execute the arrival workload has been decided before allocation instead of after allocation. The framework proposed in this paper takes both the energy consumption and performance into consideration and achieves good tradeoff between energy and consumption. The status of each hosts is controlled by the global master by regular communication, so the master can control the energy consumption and performance. Meanwhile, the computing can be done in parallel in each candidate for allocation to improve the effectiveness of computation. In summary, the main contributions of this paper are as follows:

(i) We propose the multiprocessor power model, which is shown to be close to the real power consumption of a server according to the evaluation of the model. The multiprocessor power model helps us to precisely estimate the energy consumption.

(ii) We transform the energy minimization problem of frequency scaling to a node searching problem in directed graphs. We also prove that the optimal state which consumes the least energy can be found from an initial state in which all processors' frequencies are maximum.

(iii) We provide a novel scheduling algorithm, which can work in parallel and efficiently cooperates with local DVFS controller, for the problem of energy-aware scheduling. The experiments justify the effectiveness of our strategy on energy saving.

The rest of this paper is organized as follows. Section 2 introduces some related works. Section 3 introduces the framework of our solution and Section 4 introduces the task model and the analysis of the request of a VM. In Section 5, we introduce the energy prediction method and energy minimizing algorithm of local DVFS controller. The global VM allocation algorithm is presented in Section 6. In Section 7, we evaluate our solution through some experiments. Finally, we conclude this paper in Section 8.

2. Related Work

Reducing energy consumption has already been a critical issue of data center in recent years. Many works study the energy saving strategies in virtualized environment. Kusic et al. [7] defined a dynamic resource management as a sequential optimization in virtualized environment. The sequential optimization whose objective is maximizing the profit of provider is solved using Limited Lookahead Control (LLC) by minimizing both energy cost and SLA. But the framework captures the behavior of each application by simulation-based learning and the complexity of the model makes the approach not suitable for large scale data center.

In [8], the authors have developed dynamic resource provisioning and allocation problem with virtualized technique for energy-efficient cloud computing. They propose self-manage and energy-aware mechanisms to allocate the Virtual Machines (VMs) and migrate VMs according to CPU utilizations and energy consumption. The placing problem of allocation which can be seen as a bin packing problem is solved by Modification Best Fit Decreasing (MBFD). For the migration problem, three policies are proposed to choose VMs to migrate in order to reduce energy consumption.

Cardosa et al. [9] have presented a novel approach for power-efficient VM placement for the heterogeneous data centers by leveraging min-max and share features of the VMs based on the DVFS and soft scaling technique. The power consumption and utilization obtained from the running time of a VM are optimized by being set a priori. However, their approach does not strictly support SLAs and the information of applications' priorities is needed. Cao and Dong [10] propose an energy-aware heuristic framework for VM consolidation which can obtain a better tradeoff between energy saving and performance. A SLA violation decision algorithm is proposed to determine hosts' status for SLA violation. Based on the hosts' status, the minimum power and maximum utilization policy for VM migration are used to achieve the energy saving.

Reference [11] maximizes the utilization at virtual machine level in the environment of container. The objective of the paper is to dynamically set the sizes of virtual machines in order to improve the utilization of VMs, which saves overall energy consumption. Experiments show that their method can achieve 7.55% of energy consumption compared to scenarios where the virtual machine sizes are fixed. Reference [12] proposes a VM allocation algorithm to reduce energy consumption and SLA violation, which uses the historical record of VMs' usage.

Some other works mainly focus on the DVFS strategy to decrease processors' power consumption in hosts. Some of them periodically adjust the frequency according to the performance of server. Reference [13] monitors the utilization of processors periodically and the frequency is decreased very carefully when there are observable impacts on execution time of tasks. Hsu and Feng [14] proposed a β-adaption algorithm that periodically evaluates the performance and automatically adapts the frequency and voltage at run-time. Reference [15] also developed the periodic DVFS controller for multicore processor without using any performance model.

However, the length of period has a great impact on the performance of algorithms, it should be evaluated very carefully.

Scaling the frequency according to the types of workloads is another efficient way to carry out DVFS control. They achieve the goal of energy saving with a little or limited performance loss by decreasing the frequency during the communication, data access, memory access, or idle phases. Lim et al. [16] proposed a run-time scheduler that applies DVFS control during the communication phases which is identified by intercepting the MPI calls. In [17], the authors presented a novel algorithm that utilizes the opportunities in execution of hybrid MPI/OpenMP application to scale the frequency and reduce energy consumption. Tan et al. implement the DVFS scheduling strategy for data intensive application in [18] and achieved the energy saving. Their strategy adaptively sets the suitable frequency according to the percentage of CPU-bound time in the total execution time of workloads and is implemented in source code level.

The DVFS is able to reduce the energy consumption, but it is limited on a single server. A lot of work developed the DVFS-based task scheduling among servers because the distribution of workloads influences the overall energy. References [19, 20] propose similar energy-aware strategies that schedule a set of tasks onto physical machine. They adjust supply voltage by utilizing slack time of noncritical jobs. Reference [19] also discussed the tradeoff between energy consumption and scheduling length. Khan and Ahmad [21] studied the problem of task allocation in grid and they utilized the cooperative game theory to minimize the energy consumption and makespan of tasks for DVFS-based clusters. Similar to [21], Mezmaz et al. studied the problem for the dependent precedence-constrained parallel applications [22]. Different to these works, we study the independent real-time services with deadline constraints in multiprocessor system.

References [23–27] researched energy-efficient task scheduling for real-time system. Luo and Jha studied the scheduling of periodic tasks in heterogeneous system and gave a power-efficient solution [27]. In [24], authors proposed an energy-aware task partitioning algorithm with polynomial time complexity for DVFS-based heterogeneous system. Awan and Petters proposed an energy-aware partitioning of tasks method which consists of two phases and they use a realistic power model to estimate power consumption [23]. Our task allocating algorithm cooperates with the local DVFS controller to predict the energy consumption in different situations; the influence of frequency scaling to energy consumption is taken into account before allocation for saving more energy.

3. Overview

Our framework can accept and analyze the arrival workloads and package them by Virtual Machines (VMs) and allocate them to the suitable server to reduce energy consumption. We first describe the architecture of our solution in Figure 1 and subsequently introduce the real-time analysis in this section [3]. In our solution, the *Global Scheduler* assigns a task to a VM to execute it and guarantees its QoS requirement. This VM will be allocated to a host which can offload it without causing any violation of QoS requirement and brings minimum energy consumption. Our objective is to find the allocation method for VMs and frequencies scaling method for tasks to reduce the energy consumption.

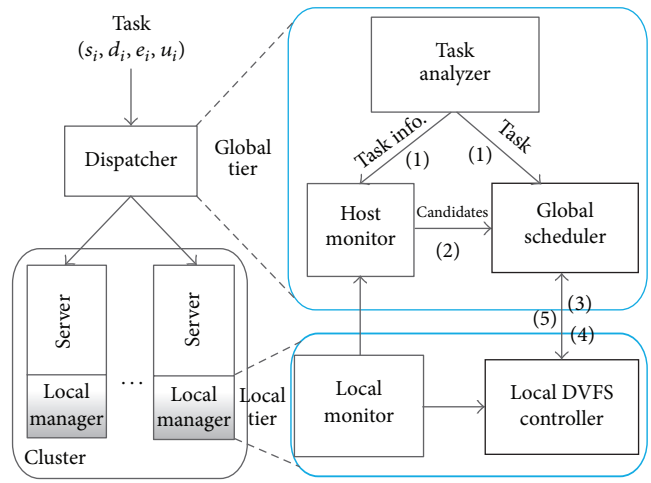


Figure 1: System architecture of our solution to the energy-aware resource management.

Definition 1 (host model). Let $\text{host}_j = (U_j, F_j)$ be denoted as resources of jth host, where U_j and F_j are vectors that record the utilizations and frequencies of each processor.

The *Task Analyzer* in Dispatcher receives and analyzes the information of incoming task and sends it to other components when necessary. The *Host Monitor* is an assistant component which connects to each server and gathers the basic information of servers. The *Local Monitor* monitors the resources of a server and sends the basic information to *Host Monitor* when necessary. The basic information of servers is recorded in the *Host Model* (Definition 1). We mainly focus on the resource of processor, so we only record the states of processors in the *Host Model*. The main work mechanism of our solution to schedule a new task request task_n is described as follows:

(1) When task_n comes, the *Task Analyzer* analyzes the basic information of task_n and sends it to the *Host Monitor* and *Global Scheduler* (Section 6).

(2) When the *Host Monitor* receives the information of task_n, it selects a set of candidates who can load task_n according to the basic information of servers and sends the set to the *Global Scheduler*. In the large datacenter, the number of candidates can be carefully selected to improve the effectiveness of allocation.

(3) When the *Global Scheduler* receives the candidates and the basic information of task_n, it sends the task information to the servers who are in the candidate set.

(4) When a candidate receives the task information, the *local DVFS controller* (Section 5) will run to estimate the minimum energy change if task_n is allocated to

one of its VM according to the monitored information. Then the controller returns the result to the *Global Scheduler*.

(5) When the *Global Scheduler* receives responses from all the candidates, it allocates task_n to the best server using our allocation algorithm. There may be some network error in communications like packet error or loss or high network delay. We can set some threshold for the *Global Scheduler*, for example, time threshold for response time or retry times. When response time or retry times of a candidate are larger than the thresholds, the *Global Scheduler* can discard this candidate.

This is the simple architecture for energy-aware task scheduling and some project implemented details or optimizations are not discussed in this paper. We mainly focus on the energy-aware scheduling for tasks and provide a solution to this problem. Some problems like single point of failure and network error are also important for the distributed cloud system. We consider that these problems have the maturing solutions in today's cloud system and these aspects may not be a problem to our solution.

4. Task Model

The request of service in the cloud computing system usually has deadline constraints which is the major aspect of Service Level Agreements (SLAs). We explore energy saving method for the cluster that accepts request for tasks. We define the *task model* (Definition 2) to describe the request for a task. The task model records some important information that users provide. s_i and d_i describe the requirements of tasks and e_i and u_i describe the execution characters of tasks.

Definition 2 (task model). Let $\text{task}_i = (s_i, d_i, e_i, u_i)$ describe ith task, where s_i, d_i, e_i, u_i represent the start time, relative deadline, predicted execution time, and the average utilization, respectively.

For the isolation, scalability, and stability of system, tasks are usually run in the VMs independently in the cloud computing system. We can regard each task as a VM, so the allocation of the tasks is equivalent to the allocation of VMs in some degree. In our model, we assign a task to a VM to run and the VM will be allocated to appropriate host. When a task finishes, the VM loading this task will be shut off or turned into sleep. The living time for a VM to run a task is equal to the execution time of this task. Therefore, the living time for the VM should not exceed the deadline of the tasks. Let VM_i represent the virtual machine load task_i.

We designed the *Task Analyzer* to accept and analyze the incoming request of tasks. It sends the basic information of tasks to other components after preprocessing. The living time of a task (i.e., VM) usually includes computing time and CPU idle time. The CPU idle time may consist of communication, memory, or disk access. The real-time analysis we designed is to distinguish the computing time and CPU idle time. The average utilization of a VM can reflect the computation and CPU idle time in some degree. Let $T_c(f)$ and T_i represent computing time at frequency f and idle time of a VM, respectively. We estimate the computing time $T_c(f_{\max}) = e_i \cdot u_i$ and idle time $T_i = e_i \cdot (1 - u_i)$ for ith task. The *Task Analyzer* calculates $T_c(f_{\max})$ and T_i and sends these information to other modules.

The computing time has a tight relation to the CPU frequency which shows a linear extension to the reduction in frequency [28, 29], while the idle time of a task will barely change due to frequency scaling. Therefore, the living time of the VM of ith task frequency f can be expressed as

$$T^i(f) = T_c(f_{\max}) \frac{f_{\max}}{f} + T_i. \tag{1}$$

When the *local DVFS controller* predicts the energy consumption in different frequency, the living time of VMs can be calculated by (1) according to the task information provided by Task Analyzer. Although the running time can be predicted under different frequency, the energy prediction and DVFS controller are not a easy task. We will introduce details of our method to solve them in next sections.

5. Local DVFS Controller

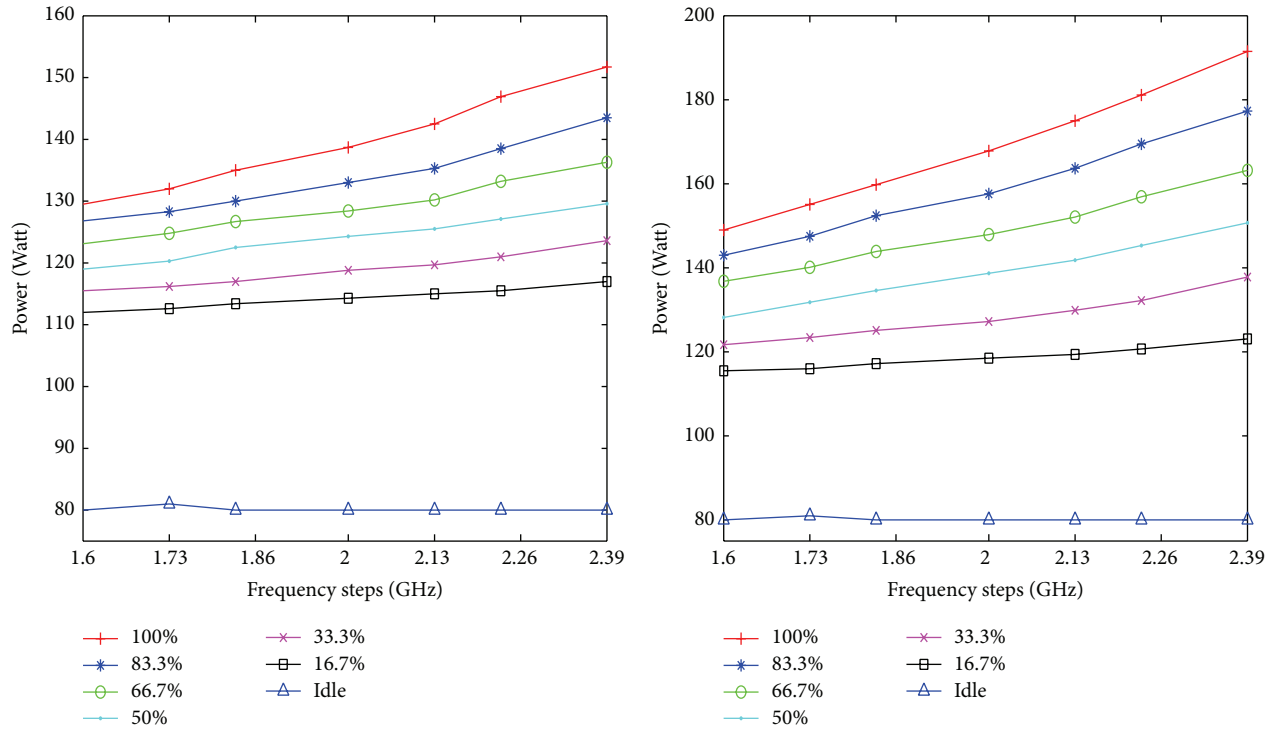
The *DVFS Controller* plays an important role in our framework and it has two main functions. On the one hand, it predicts the energy consumption of a multiprocessor server according to the processors' utilizations, frequencies, and the living time of VMs. On the other hand, based on the energy prediction, it runs the *k-Phase energy Prediction (kPP)* algorithm to find the best frequencies combinations that bring minimum energy consumption.

5.1. Energy Prediction for Multiprocessor Servers. The electric energy consumption is the integral of the active power with respect to time. Therefore, the power prediction of server is crucial. Previous works like [30–34] provided serval methods to estimate the power of a server. However, they only focused on the frequency-power or utilization-power relationship and the detailed power prediction for multiprocessor platform is also ignored. In this paper, we provide a practical power prediction for multiprocessor servers based on the frequency-power and utilization-power relationship. We utilize the fact that the homogenous processors will consume the same power when they are under the same condition to predict the power consumption.

The power consumption of a server consists of two parts: static and dynamic power consumption. The static parts include the power consumption of main board, hard disk, fan, and so forth. CPU accounts for the largest part of dynamic power. According to the previous studies, the dynamic power consumption of CPU is proportional to the frequency and to the square of voltage [34], which can be express as

$$P_{\text{dynamic}} \simeq A \times C \times V^2 \times f, \tag{2}$$

where A is the percentage of active gates, C is total capacitance, V is supply voltage, and f is the operating frequency. According to [31], the voltage has a linear relationship to



(a) Only one processor is busy

(b) Two processors are busy

FIGURE 2: Real power consumption of Dell R710 whose configurations are introduced in Table 1 and the legend represents the different utilizations.

frequency, so the dynamic power of a processor can be reduced as a function of frequency: $P_{\text{dynamic}} = \alpha \times f^3$, where α is a proportional coefficient. Processors also have static power when they are active. Let P_s represent the static power of a server and P_{CPU_s} represent the static power of processor. The power of a host in which all homogenous processors work in the same frequency f with full utilization can be expressed as follows:

$$P(f) = P_s + N_c\left(P_{\text{CPU}_s} + \alpha f^3\right), \tag{3}$$

where N_c is the number of CPUs. We want to eliminate the static power of processors, which is not easy to measure. For a given host, we can easily measure its maximum power which is $P_{\max} = P_s + N_c(P_{\text{CPU}_s} + \alpha f_{\max}^3)$. Therefore, we can estimate the power consumption of a host in which all processors work in the same frequency f:

$$P(f) = P_{\max} - \alpha N_c\left(f_{\max}^3 - f^3\right). \tag{4}$$

The power consumption is also related to utilizations. Figure 2(a) shows the power consumption with only one processor running and Figure 2(b) shows the power of two processors that work in same utilization and frequency. As we can see, the power with different utilization under same frequency is different. The power and the utilization present a linear relationship which is with one voice to [30, 32, 33]. Therefore, the power consumption of one homogenous CPU with frequency f and utilization u can be denoted as

$$P_{\text{CPU}}(u, f) = \frac{1}{N_c}\left[P_{\max} - P_s - \alpha N_c\left(f_{\max}^3 - f^3\right)\right]u. \tag{5}$$

Finally, the power of prediction of a homogenous multiprocessor server can be expressed as

$$\begin{aligned} P_{\text{host}} &= P_s + \sum_{c=1}^{N_c} P_{\text{CPU}}^c \\ &= P_s \\ &\quad + \frac{1}{N_c}\sum_{c=1}^{N_c}\left[P_{\max} - P_s - \alpha N_c\left(f_{\max}^3 - F_{j,c}^3\right)\right]U_{j,c}. \end{aligned} \tag{6}$$

We can view the power of jth host as a function of utilizations and frequencies, which is presented as $P_{\text{host}}(F_j, U_j)$, where U_j and F_j are defined in *Host Model*.

Definition 3 (Frequency Scaling Unit). A time interval $[t_1, t_2)$ is a Frequency Scaling Unit (FSU) if (1) $\forall t \in [t_1, t_2)$, $NT(t) = NT(t_1)$ and (2) $NT(\cdot)$ changes at t_1 and t_2, where $NT(t)$ represents the number of VMs at time t in a server.

The energy consumption depends on both the execution time and the power. We define the concept of *Frequency Scaling Unit* (FSU, Definition 3) to estimate the living time

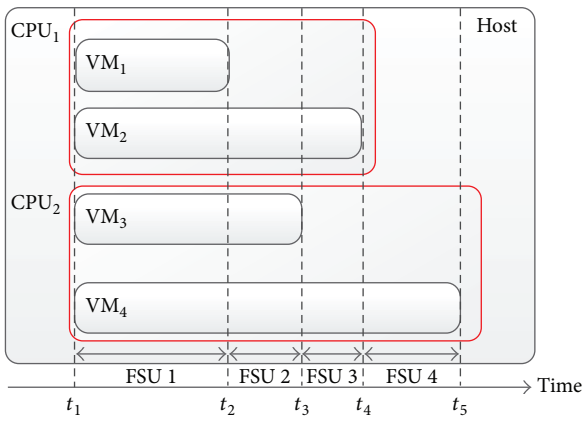


FIGURE 3: An example of FSU.

of VMs. The FSU represents a period of time that the number of VMs does not change. Once the number of VMs changes, that is, a VM coming or leaving, it enters the next FSU. An example of FSU is shown in Figure 3, which includes four FSUs. Assuming a VM is stopped at time t_2 and next VM is ended at time t_3, then $T = t_3 - t_2$ is an FSU. If a VM is allocated to the server at t_1 and the VM finished at t_2, $T = t_2 - t_1$ is said to the *first FSU* from current time. It is obvious that the number of VMs in the host is equal to the number of FSUs if all VMs finish at the different time, and we suppose that VMs are ended at the different time in a host in the rest of this paper.

If we set consistent frequencies for all processors in an FSU, the power state in this FSU is relatively stable because the workloads in this FSU are fixed. We know the length of this FSU, so the energy consumption in an FSU can be predicted conveniently and precisely by the following equation: $E = P \times T$, where P and T represent power and time, respectively. Based on the definition of FSU, power function, and related notations in Notations, the energy consumption of jth host to finish all the VMs can be predicted as follows:

$$E_j = \sum_{k=1}^{N_{j,p}} P_{j,k}\left(U_{j,k}, F_{j,k}\right) T_{j,k}, \tag{7}$$

where the power of host $P_{j,k}$ can be calculated by (6) in different situations. The length of FSU can also be estimated under different frequencies by (1).

5.2. kPP Algorithm for Energy Minimization. According to the analysis of energy prediction, if we set consistent frequencies in an FSU, then we can predict the energy consumption in an FSU conveniently. If we set FSUs with different frequencies, the living time of VMs and power state of server will be different, which brings different energy consumption. There is an optimal solution that consumes minimum energy when all VMs end in this server. We want to find the frequencies combinations for all FSUs that bring minimum energy on the promise of ensuring the requirements of VMs. Based on the energy prediction, the energy minimization problem of frequency scaling in homogenous multiprocessor platforms can be formalized as follows:

$$\min_{F_{j,k} \in F_j} \sum_{k=1}^{N_{j,p}} P_{j,k}\left(U_{j,k}, F_{j,k}\right) T_{j,k}$$
$$\text{s.t.} \quad \forall j \in H,\ \mathrm{VM}_i \in J_j,\ s_i + \sum_{m=1}^{N_p^i} t_{j,m}^i < d_i. \tag{8}$$

For clearly describing the problem, we define $(F_{j,1}, F_{j,2}, \ldots, F_{j,|J_j|})$ as a *state* of possible frequencies combinations for FSU 1 to $|J_j|$ using the notations in Notations. If there is only one frequencies combination that is different between two states, we say they are neighbors. For example, if there are two states $s_1 : (F_{j,1}, F_{j,2}, \ldots, F_{j,|J_j|})$ and $s_2 : (F_{j,1}, F_{j,2}, \ldots, F'_{j,|J_j|})$ and the frequencies combinations of FSU $|J_j|$ in s_1 and s_2 are different while others are the same, then s_1 and s_2 are neighbors. In addition, we define $E(s)$ as the cost function of total energy consumption of s according to (7) if we scale the frequencies like s in each FSU. Let a node present a state and an edge (u, v) between two nodes presents neighborhood between u and v. The minimization problem is to find the "optimal" node that brings minimum energy without any violation of SLAs from the initial node in the graph.

Lemma 4. *Let the initial node represent the state in which all processors' frequencies are highest in all FSUs. If the initial state ensures SLAs for all VMs, there is a path from initial node to the optimal node with minimum energy consumption without any violation of SLAs.*

Proof. Let $s_m = (F_{j,1}, F_{j,2}, \ldots, F_{j,|J_j|})$ represent the optimal state with minimum energy consumption without any violation of SLAs. If all processors' frequencies are highest in all FSU of s_m, the initial state is the optimal state. Otherwise, we select the FSU r in which the frequencies are not highest for all CPUs. If we scale the frequencies to highest in r, the new state s_n will also ensure the SLAs for all VMs because processors are working at higher frequencies which leads to shorter execution time. s_n is one of the neighbors of the optimal state, which means that s_n can also move to s_m. Repeating the process above, we can find a path from s_m to initial state s_i, which represents that there is a path from s_i to s_m. □

5.2.1. k-Phase Energy Prediction (kPP) Algorithm. By energy prediction of k FSUs, the best frequencies combinations can be found moving from the initial state according to Lemma 4. There are $|F_j|$ possible frequencies combinations in each FSU, so there may be $|F_j|^{|J_j|}$ possible states of all FSU with different energy consumption. Let FL_j represent the frequency levels of jth host; we have $|F_j| = |FL_j|^{|C_j|}$. We want to find the optimal solution with minimum energy consumption in these $|F_j|^{|J_j|}$ possible states while still ensuring SLAs. However, the searching space may be $|FL_j|^{|C_j||J_j|}$ which is too huge if there are many VMs. Therefore, we provide two heuristic algorithms to search the "optimal" solutions which are based on simulated annealing (SA) [35] and variable depth search (VDS) [36], respectively.

Input:
The state of host$_j$;
Output:
The possible state s;
(1) J_j = host$_j$.getVM(), F_j = host$_j$.getFreqSpace()
(2) set s_0 be the state that the frequency is max for each CPU in all FSU
(3) $s = s_0$, $e = e_{\max} = E(s_0)$, $t = 0$, $k = |J_j|$
(4) **while** $t < t_{\max}$ or s doesn't change in l rounds **do**
(5) $st = s$, r = random(k)
(6) $F_{j,r}$ = random($F_j - s$.get(r)) /*Select a neighbor*/
(7) st.set($r, F_{j,r}$) /*Change frequencies combination of FSU r to $F_{j,r}$*/
(8) **for** $i = 1$ to k **do**
(9) **if** VM$_i$ violates SLAs according to st **then**
(10) go to (17)
(11) **end if**
(12) **end for**
(13) $et = E(st)$
(14) **if** $et \le e_{\max}$ or random() $< \exp(-(et - e)t/pT)$ **then**
(15) $s = st$, $e = et$ /*Change states*/
(16) **end if**
(17) $t = t + 1$
(18) **end while**
(19) **return** s

Algorithm 1: SA based kPP algorithm.

(1) Simulated Annealing Based Heuristic Algorithm. By comparing energy consumption of a random neighbor, we can find a better state that brings less energy. If we repeat the process many times, we may find the optimal state. Let s_0 represent the initial state in Lemma 4. In fact, since the simulated annealing (SA) algorithm has been proved to converge to the optimum with probability 1, it can be expected that our algorithm will output nice results by enough iterations. If we know the frequency steps and tasks' information, the living time of VMs is determined. Therefore, we can estimate the total energy of the situation of state s_0 (line 3) using the energy cost function $E(\cdot)$. The algorithm runs $t_{\max}$ iterations to find the state where less energy is consumed compared to the initial state. In each iteration, the algorithm randomly selects an FSU r to change the frequencies of processors and generates a new state st. This step takes $O(1)$ time. If the random neighbor st violates SLAs for any one of VMs, the state is discarded and our algorithm enters into the next iteration. This step takes $O(N_{j,p})$ time, where $N_{j,p}$ is the number of tasks. Otherwise, if predicted energy et is less than e, st is selected as compared state for next iteration due to less energy consumption. The energy prediction takes $O(N_{j,p} \cdot N_c)$ time according to (7), where N_c is the number of processors. Besides, the algorithm also changes the state from s to st with the probability $\exp(-(et - e)/pT)$ suggested by Metropolis et al. [37] to give the possible to find optimal solution. The details of the simulated annealing based kPP algorithm are presented in Algorithm 1. Obviously, the time complexity of SA-based kPP algorithm is $O(N_{j,p} \cdot N_c \cdot t_{\max})$.

(2) Variable Depth Search Based Heuristic Algorithm. The VDS-based kPP algorithm selects the state that brings minimum energy in a subset of neighbors and compares it to the current state. If the selected state consumes less energy on the promise of ensuring the SLAs of VMs, we will change the state to it. The initialized state of VDS-based algorithm is the same as the initialization of SA-based algorithm. The algorithm selects a subset of neighbors whose frequencies combination of FSU r are different (lines 5-6). The frequencies combination of FSU r with minimum energy will be selected (line 7) and generates a new state. The energy prediction takes $O(N_{j,p} \cdot N_c)$ time, so the selection of state with minimum energy takes $O(|X| \cdot N_{j,p} \cdot N_c)$ time, where $|X|$ is the size of subset. The algorithm checks the violations of SLAs of new state (lines 9–13). The process repeats for $t_{\max}$ times or until the state s does not change in l iterations. Therefore, the time complexity of this algorithm is $O(|X| \cdot N_{j,p} \cdot N_c \cdot t_{\max})$. The effectiveness of the variable depth search is proved in [36]. The details of VDS-based kPP algorithm are shown in Algorithm 2.

The *local DVFS controller* runs the kPP algorithm when the *Global Scheduler* asks it to predict the minimum energy consumption and return it to *Global Scheduler*. This is one of the opportunities to run kPP algorithm. When the workload changes, the power state will change. In addition, the execution time of VMs may have some errors which may lead to the error of energy prediction. Therefore, we apply the frequencies scaling when a VM finishes and scale the frequency for first FSU, which means that the algorithm only scales the CPUs' frequencies just for the first FSU while predicting the frequencies combinations for k FSUs. As the example shown in Figure 3, if the VM$_4$ comes at the time t_1 and is allocated to the host, this host applies the kPP algorithm at that time and sets the CPUs' frequencies like FSU 1 of the result. When

Input:
The state of host$_j$;
Output:
The possible state s;
(1) J_j = host$_j$.getVM(), F_j = host$_j$.getFreqSpace()
(2) set s_0 be the state that the frequency is max for each CPU in all FSU
(3) $s = s_0$, $t = 0$, $k = |J_j|$
(4) **while** $t < t_{\max}$ or s doesn't change in l rounds **do**
(5) $st = s$, r = random(k)
(6) randomly select a subset $X \subseteq F_j$
(7) x = argmin($E(st.\text{set}(r, x))$), for all $x \in X$ /*Select the state form subset with minimum energy consumption*/
(8) st.set(r, x) /*Change frequencies combination of FSU r to x*/
(9) **for** i = 1 to k **do**
(10) **if** VM$_i$ violates SLAs according to st **then**
(11) go to (15)
(12) **end if**
(13) **end for**
(14) $s = st$ /*Change states*/
(15) $t = t + 1$
(16) **end while**
(17) **return** s

Algorithm 2: VDS-based kPP algorithm.

VM$_1$ finishes at the t_2, the kPP algorithm also runs to obtain the "optimal" state s and scales frequencies according to the result. Due to the specialities of kPP algorithm at running time, the iteration should be completed in a short time so that the local stage can scale the frequencies in time.

6. Global Scheduler

Different allocations of a new VM may affect the overall energy consumption, because the new VM executed on different servers will bring different energy consumption. We want to find the appropriate scheduling to minimize the energy consumption to finish all the VMs. We can obtain the different energy consumption with different allocation if we ask each host to predict the minimum energy consumption. Using the results of different allocations, we can select a better allocating scheme to reduce the energy consumption. Our goal is to minimize the overall energy cost of the whole cluster for finishing all VMs including the new VM VM$_n$. To solve the energy minimization problem of VM scheduling, we first formalize the problem. Let decision parameter $a^i_{j,c}$ be 1 if the ith VM is allocated to cth CPU of jth host, otherwise 0. The energy-efficient VM allocation problem can be expressed as

$$\begin{aligned}
\min \quad & \sum_{j\in H}\left\{\min_{F_{j,k}\in F_j}\sum_{k=1}^{N_{j,p}} P_{j,k}\left(U_{j,k}, F_{j,k}\right)T_{j,k}\right\}\\
\text{s.t.} \quad & \sum_{j\in H, c\in C_j} a^n_{j,c} \le 1;\\
& J_j = J_j \cup \{\text{VM}_n\}, \quad \text{if } a^n_{j,c} = 1;\\
& \forall j \in H,\ \text{VM}_i \in J_j,\ s_i + \sum_{m=1}^{N^i_p} t^i_{j,m} < d_i;\\
& \forall k \in N_{j,p},\ j \in H,\ \sum_{c\in C_j} U^c_{j,k} < UT_j.
\end{aligned} \tag{9}$$

The energy minimization problem of VM scheduling is to find the server which brings minimum energy of whole cluster if VM$_n$ is allocated to it. The minimum energy consumption of each server can be predicted by kPP algorithm, represented by EMIN$_j$ for jth host. If an incoming VM is allocated to jth host, the value of EMIN$_j$ changes while the minimum energy consumption of other hosts does not change. When VM$_n$ is allocated to the yth host, the energy consumption becomes EMIN$'_y$ + $\sum_{j\in H-\{y\}}$ EMIN$_j$, where EMIN$'_y$ is the minimum energy cost if VM$_n$ is allocated to yth host. We have

$$\begin{aligned}
E_{\min} &= \text{EMIN}'_y + \sum_{j\in H-\{y\}} \text{EMIN}_j\\
&= \Delta\text{EMIN}_y + \text{EMIN}_y + \sum_{j\in H-\{y\}} \text{EMIN}_j\\
&= \Delta\text{EMIN}_y + \sum_{j\in H} \text{EMIN}_j.
\end{aligned} \tag{10}$$

So we can select the host that brings minimum energy change ΔEMIN$_y$ to run the incoming VM. We call the scheduling algorithm *Minimum energy Change* (MC), shown in Algorithm 3. The *Global Scheduler* sends the information of a VM after analyzing to a subset of host (line 1) and each host returns the predicted minimum energy change on it. Therefore, we can run the energy-efficient algorithm in parallel to

Input:
A new VM VM_n;
Output:
Designate host and processor for loading VM_n;
(1) *Host Monitor* selects a subset of active hosts that can load the VM
(2) notify the information of VM_n to all candidates
(3) each host estimates the minimum energy change $\Delta EMIN_j$ if VM is allocated to processor c of $host_j$
(4) $host_j = argmin_{j \in H}(\Delta EMIN_j)$
(5) **return** $host_j$

Algorithm 3: Minimum energy change allocation.

obtain the minimum energy consumption for each host when a VM arrives. After the *local DVFS controller* predicts the minimum energy change, it also records the best processor to hold this VM. When this VM is really allocated to it, the VM will be scheduled onto this best processor. Once deciding the host, the selected host will start a VM to run the VM working under the selected frequencies. It is obvious that the time complexity is $O(|H|+L+T)$, where L and T represent the time complexity of local predicting algorithm and communication time, respectively.

The number of candidate hosts will affect the total cost of a cluster, we evaluate the influence of kPP strategy on the total energy cost. Assume the total VM number is N_t; the size of subset for candidate hosts is N_h and the average run-time of local DVFS algorithm is 0.5 seconds. Let the mean power consumed by a VM be $\overline{P}$ Watt and the average length of VMs be $\bar{t}$ seconds. The kPP algorithm runs when the MC algorithm asks candidate hosts to estimate energy consumption; the energy consumption of kPP algorithm in this part is $N_h \cdot \overline{P} \cdot N_t \cdot 0.5$. The kPP algorithm also runs when a VM is allocated and finished, so the energy for this part is $2 \cdot N_t \cdot \overline{P} \cdot 0.5$, and the total energy produced by all VMs is $N_t \cdot \overline{P} \cdot \bar{t}$. Therefore, the *energy consumption ratio* (ECR) of kPP algorithm compared to the total energy cost is

$$\mathrm{ECR} = \frac{0.5 \cdot N_h + 1}{\bar{t}}. \tag{11}$$

In a large scale data center, the mean VM length can be acquired according to the historical data, and we can carefully select the size of candidate host estimating the energy change of offloading a new VM to increase the energy consumption of kPP algorithm as less as possible.

7. Experimental Evaluation

7.1. Evaluating Power Prediction. The energy prediction of server depends on the accuracy of power prediction under different utilizations and frequencies. We have evaluated the multiprocessor power prediction method by comparing the real power consumption to the estimation of power model in different status for a specific host. The real experimental environment is shown in Figure 4. The details of server R710 used in our paper are shown in Table 1. We explore the real power consumption R710 and use the first seven steps when

Table 1: Details of servers.

Name	Dell PowerEdge R710	Dell PowerEdge R720
CPUs	Two Intel Xeon processors E5645 @2.4 GHz	Two Intel Xeon processors E5-2620 @2.1 GHz
Frequency steps (GHz)	1.60, 1.73, 1.86, 2.00, 2.13, 2.26, 2.39, 2.40	1.20, 1.30, 1.40, 1.50, 1.60, 1.70, 1.80, 1.90, 2.00, 2.10
Memory	24 G 1333 Mhz DDR3	64 G ECC DDR3
Disk	Two 10 k SAS, 250 GB	One 10 k SAS, 300 GB
Operation system	CentOS 6.5	CentOS 6.5

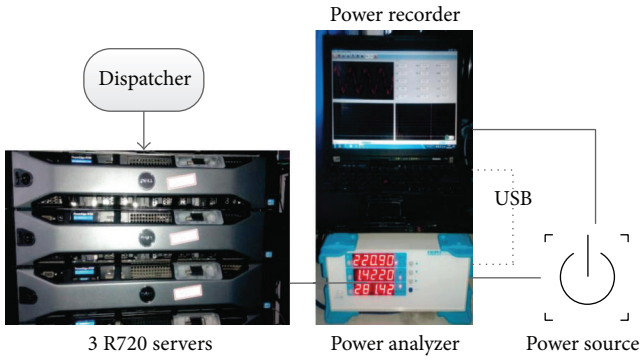


Figure 4: Real system architecture.

we evaluate the power model because the last frequency is very close to the frequency 2.39 GHz. The power consumption of the host when both processors are fully utilized at frequency level 2.39 GHz is 192 Watt and the static power when the system is not idle is 110 Watt. The proportional coefficient $\alpha = 2.33135$ is obtained and calibrated by offline experiments.

For evaluating the multiprocessor power prediction, we randomly select some frequencies and utilizations of two processors and use power model to estimate the power consumption. At the same time, we measure the real power consumption of R710 server with the same frequencies and utilizations of processors and results are shown in Table 2. We use the Aitek AWE 2101 power analyzer to measure power. Table 2 shows that the estimated power is very close to the real

Table 2: Power comparison.

Frequencies (GHz)		Utilizations		Power (Watt)		Error
CPU_1	CPU_2	CPU_1	CPU_2	Estimation	Real	
1.60	2.00	61%	87%	145.33	147.01	−1.14%
1.73	2.13	95%	84%	156.45	157.81	−0.86%
1.86	1.73	19%	14%	117.50	117.03	+0.40%
2.00	1.60	59%	9%	127.96	128.32	−0.28%
2.13	2.13	61%	84%	155.67	155.68	+0.00%
2.13	1.86	58%	57%	141.94	142.68	−0.52%
2.26	2.00	58%	30%	139.10	137.89	+0.87%
2.26	2.39	98%	81%	178.21	181.69	+1.91%
2.39	1.60	92%	18%	150.87	149.79	+0.72%
2.39	2.13	12%	59%	133.48	133.15	+0.25%

Table 3: Run-time versus SA iterations.

SA iterations	2 CPUs with 12 VMs		4 CPUs with 24 VMs	
	RT (ms)	EC (J)	RT (ms)	EC (J)
0	0	2466027	0	3210277
10	1	2397905	1	3195957
100	3	2385065	7	3117021
1000	20	2323755	50	3004518
10000	170	2301687	486	2952581
100000	1676	2293194	4895	2923625
1000000	16645	2289410	46070	2851156

Table 4: Run-time versus VDS iterations.

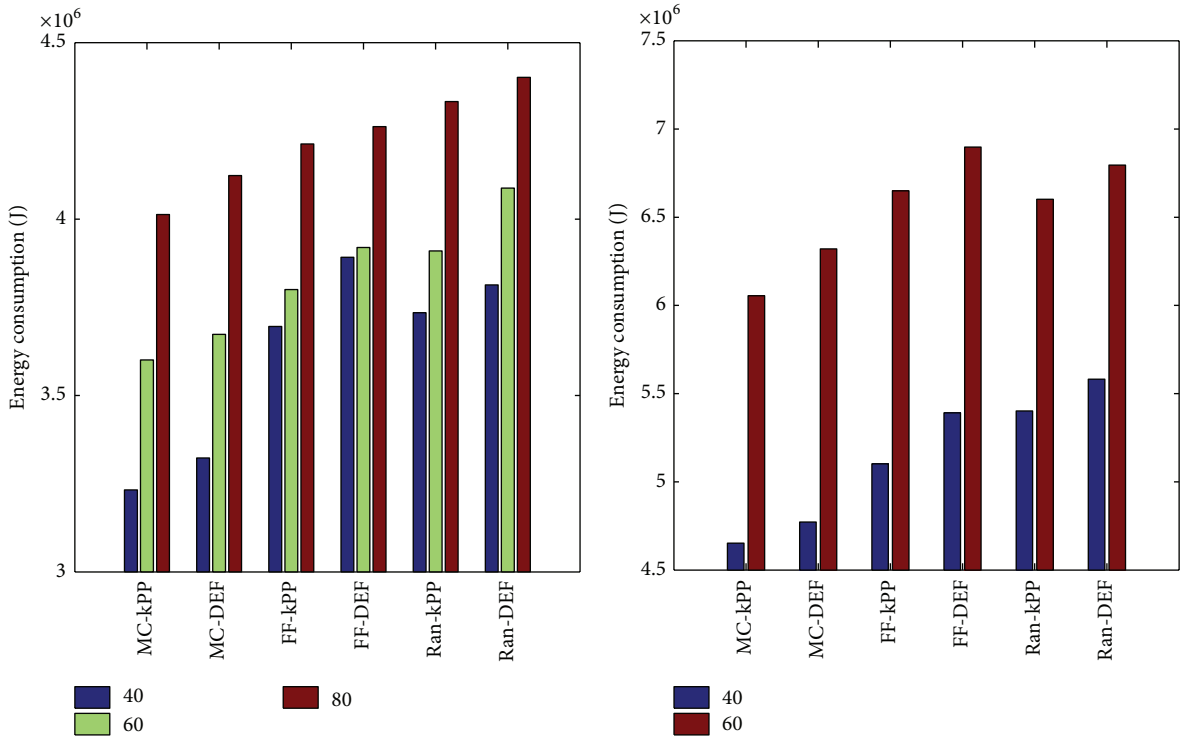
VDS iterations	2 CPUs with 12 VMs		4 CPUs with 24 VMs	
	RT (ms)	EC (J)	RT (ms)	EC (J)
0	0	2466027	0	3210277
10	5	2379716	12	3080983
100	38	2302226	118	2926700
1000	361	2293796	1145	2859979
10000	3519	2289368	11682	2855693
100000	35245	2289438	115659	2843434
1000000	349465	2289410	1140554	2848026

power consumption of the server with the same utilizations and frequencies of different processors.

7.2. Convergence Speed. In this subsection, we compare the convergence speeds of the two algorithms and present them in Tables 3 and 4. The reported run-times and iteration times are for running the two algorithms of a synthetic 2-processor and 4-processor with 7 frequency levels machine where 12 and 24 VMs are executed in parallel. The RT and EC in Tables 3 and 4 represent running time and energy consumption, respectively. We can draw three obvious conclusions: (1) the SA-based algorithm iterates significantly faster than the VDS-based algorithm which means that more iterations can be executed during the same period; (2) the VDS-based algorithm outperforms the SA-based algorithm while it leads to the same iteration times if the host is equipped with serval processors; (3) when the processor number and frequency levels are relatively small, both the two algorithms converge rapidly and obtain the close results. And the VDS-based algorithm perform better than SA-based algorithm when the number of processors is small. This conclusion suggests that the service provider may prefer the SA-based algorithm if they persist in finding the best frequency configurations.

Notice that, in this experiment, we use a very extreme setup where a 4-processor host is enforced to run as much as 24 VMs at the same time, which means a processor must be responsible for 6 VMs on average. In fact, in the real datacenters, it can be expected that the average VM number on a single processor is far less than 6. Thus our algorithm can run efficiently enough to serve for our online VM scheduling algorithm and obtain an accepted result within 1 second which is close to results of more iterations.

7.3. Experiments in Real Environment. Our real experimental environment has three servers and a controller on a virtual machine. The power is measured by Aitek Power Analyzer AWE2101. Each R720 server whose details are shown in Table 1 runs the kPP algorithm to predict energy and control processors' speed. We combine the kPP algorithm with the random (Ran) and first-fit (FF) VM scheduling. The *random* scheme allocates the coming task to the processor randomly from the subset of processors which can offload the new task without causing any violations of QoS requirements. The *first-fit* scheme gives each processor an index and allocates the coming task to the processor with smallest index who can offload the new task without causing any violations of QoS requirements. Meanwhile, the proposed global assignment (MC) is also combined with the default DVFS controller Ondemand [38] (DEF) in Linux. The two-tier energy-aware resource management proposed in this paper is represented by *MC-kPP*. We compare these six strategies to evaluate the performance of our solution on energy savings. For each VM, its execution time is generated uniformly at random between a *minimum* and *maximum* living time represented by $ET_{\min}$ and $ET_{\max}$, respectively. The deadline of a VM is set from 1 to 1.5 times longer to its execution time randomly. Moreover,



(a) $ET_{\min} = 600, ET_{\max} = 3600$ (b) $ET_{\min} = 600, ET_{\max} = 7200$

Figure 5: Energy consumption of real system. The legend in (a) and (b) means that the number of VMs needs to be allocated.

their utilizations requirements follow the normal distribution with $\mu = 0.75$ and $\sigma = 1$. In addition, the arriving times of VMs follow a Poisson distribution with different average rates.

In these experiments, the iteration times are 10000 for SA-based kPP and 1000 for VDS-based kPP and the number of neighbors in VDS-based kPP algorithm is 20. The results of SA-based and VDS-based algorithms are very close, so we show the results of VDS-based kPP algorithm in Figure 5 whose legend represents different numbers of VMs. Meanwhile, the size of candidates in MC algorithm is equal to the number of servers. As we can see in Figure 5, the energy savings of our solution can reach from 8% to 17% in the real environment with 3 servers.

7.4. Simulation Results. Due to the inaccessibility of a large scale datacenter, we conduct the simulations to evaluate MC-kPP solution in a larger cluster. We model Dell R710 servers to service the dynamically arriving VMs. Meanwhile, the attributes of generated VMs are the same as the attributes introduced in Section 7.3.

As we can see in Figure 6(a), the local kPP algorithm can reduce energy consumption of a specific server compared to Ondemand strategy. In addition, the influences of global scheduling algorithm are greater than the influences of local DVFS controller on energy savings when different scheduling algorithms are applied. The lengths of VMs in Figure 6(b) are generated uniformly and randomly between 600 and 7200 seconds. The legend in Figure 6(b) represents the arriving ratio of VMs in one minute. The results show an increasing tendency of the energy saving ratio with the increments of VM numbers and the best result can reach about 28%. In Figure 6(c), we investigate the influence of lengths of VMs; VMs are generated in different lengths which are shown in the legend. As shown in Figure 6(c), MC-kPP can also save more energy when the VMs become more. At the same time, MC-kPP performs better when the average execution time of VMs becomes longer, because the influence of local kPP algorithm itself becomes smaller and the effectiveness of frequencies scaling becomes more obvious. In addition, the size of subset in MC algorithm is also investigated in Figure 6(d); the energy consumption of kPP algorithm in local machine is below 0.5% of total energy consumption when the average execution time is long. With the increment of subset size, the performance of MC-kPP is improved because a better server can be found in a larger scale. We also evaluate the effectiveness of MC-kPP in different scales of data centers ranging from 50 to 5000 servers with different features of arriving VMs. According to the results of Figure 7, the MC-kPP outperforms other strategies in different scales of datacenters, which can reach about 25% energy savings. With the increasement of host numbers and VM numbers, MC-kPP performs stably in different scenarios.

8. Conclusions

In this paper, we propose a cooperative two-tier energy-efficient strategy to manage the VM allocations and adapt

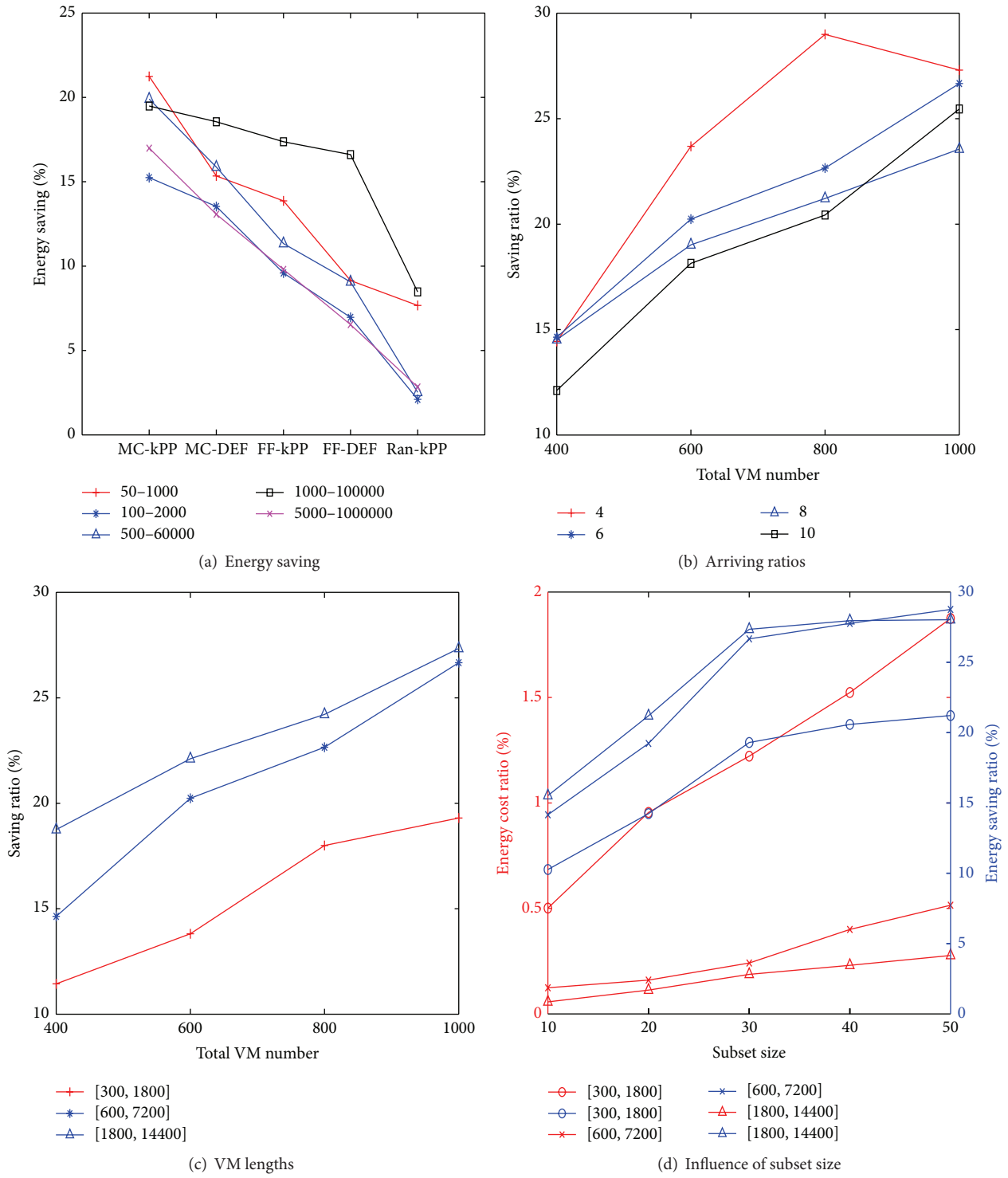


Figure 6: Performance evaluation on different aspects. The legend in (a) represents the "server numbers-total VM number." The legend in (b) represents the "VM request number arriving in a minute." The legend in (c) and (d) represents the "(minimum living time, maximum living time)."

frequencies scaling for saving energy. A frequency scaling algorithm is proposed based on the practical power and energy prediction. The Global Scheduler collaborates with local DVFS controller to assign VMs and save overall energy. In addition, two heuristic algorithms are provided for searching the optimal solutions which predict minimum energy consumption. The time complexities of both the algorithms are acceptable with satisfactory results according to the

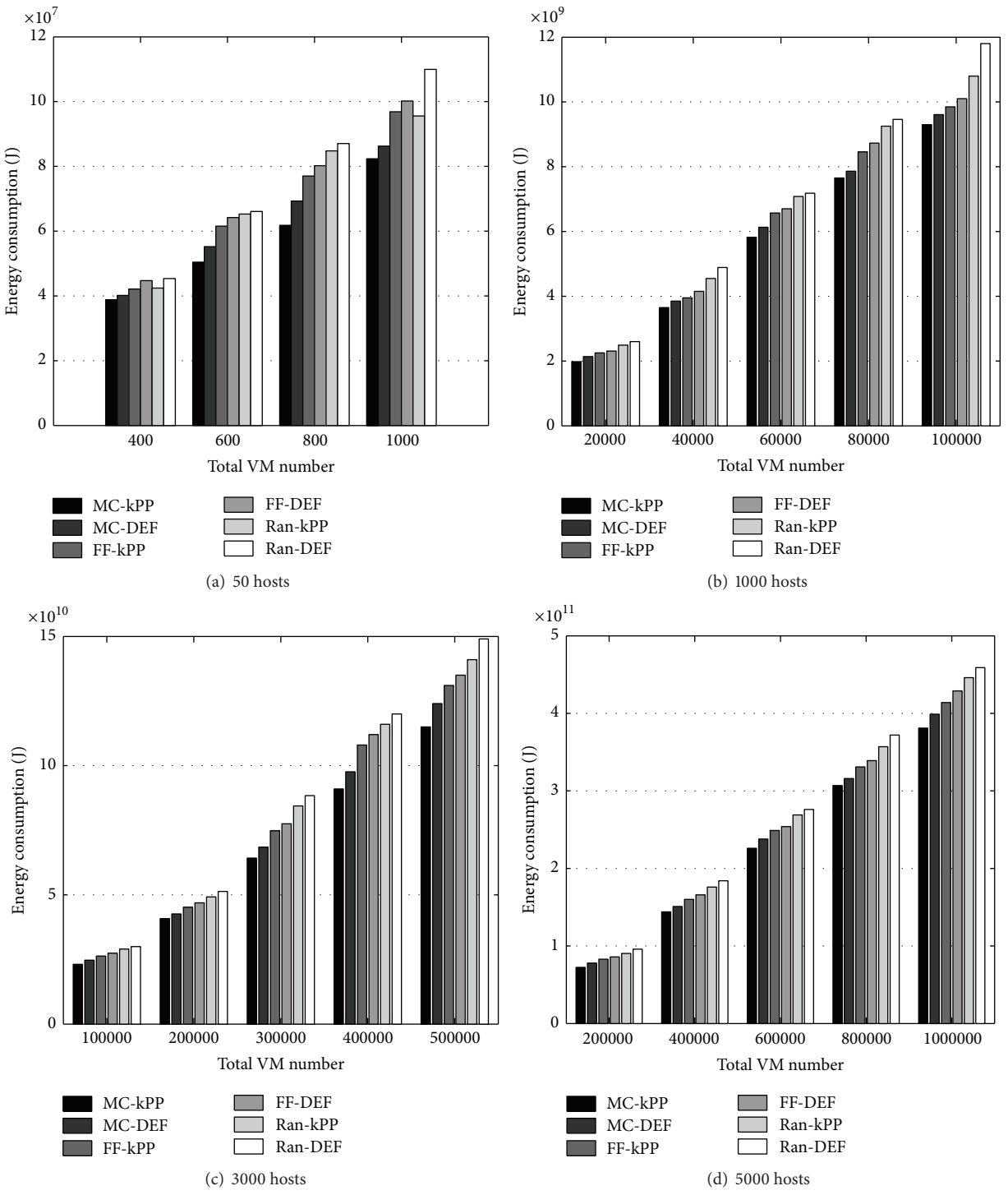


Figure 7: Energy consumption of different scale of datacenters with different number of VMs.

experiments. Finally, the real experiment results justify the effectiveness of MC-kPP.

Notations

J_j: The set of jobs (VMs) in jth host
C_j: CPU set of jth host
F_j: All possible combinations of frequencies for CPUs on jth host
$F_{j,k}$: Combinations of frequencies for CPUs on jth host in kth
$U_{j,k}$: Utilizations of CPUs of jth host in kth FSU
$U^c_{j,k}$: Utilization of cth CPU of jth host in kth FSU
$T_{j,k}$: Time length of kth FSU of jth host

$N_{j,p}$: The number of FSUs from the current time of jth host

$P_{j,k}(\cdot)$: The power function of jth host in kth FSU.

Disclosure

This is a substantially extended version of the paper presented at ICA3PP 2015 [3].

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under Grants nos. 61472181, 61100197, and 61202113; Jiangsu College Natural Science Foundation under Grant no. 14KJB520016; Jiangsu Natural Science Foundation under Grant no. BK20151392; and JSPS KAKENHI Grant no. 16K00117. And this work is also partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] W. Forrest, "How to cut data centre carbon emissions?" 2008.

[2] A. Beloglazov, R. Buyya, Y. C. Lee et al., "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers*, vol. 82, no. 2, pp. 47–111, 2011.

[3] W. Huang, J. Shi, Z. Wang, and Z. Qian, "BiTEM: a two-tier energy efficient resource management framework for real-time tasks in clusters," in *Algorithms and Architectures for Parallel Processing*, G. Wang, A. Zomaya, G. M. Perez, and K. Li, Eds., vol. 9529 of *Lecture Notes in Computer Science*, pp. 494–508, Springer, Berlin, Germany, 2015.

[4] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware '08)*, pp. 243–264, Springer, Leuven, Belgium, December 2008.

[5] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No power struggles: coordinated multi-level power management for the data center," *ACM SIGARCH Computer Architecture News*, vol. 36, pp. 48–59, 2008.

[6] B. Guenter, N. Jain, and C. Williams, "Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning," in *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM '11)*, pp. 1332–1340, Shanghai, China, April 2011.

[7] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[8] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges," https://arxiv.org/abs/1006.0308.

[9] M. Cardosa, M. R. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM '09)*, pp. 327–334, June 2009.

[10] Z. Cao and S. Dong, "An energy-aware heuristic framework for virtual machine consolidation in Cloud computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 429–451, 2014.

[11] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Efficient virtual machine sizing for hosting containers as a service (SERVICES 2015)," in *Proceedings of the IEEE World Congress on Services (SERVICES '15)*, pp. 31–38, New York, NY, USA, June-July 2015.

[12] Z. Zhou, Z. Hu, and K. Li, "Virtual machine placement algorithm for both energy-awareness and sla violation reduction in cloud data centers," *Scientific Programming*, vol. 2016, Article ID 5612039, 11 pages, 2016.

[13] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," in *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '02)*, pp. 356–367, IEEE, November 2002.

[14] C.-H. Hsu and W.-C. Feng, "A power-aware run-time system for high-performance computing," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC '05)*, IEEE Computer Society, Seattle, Wash, USA, November 2005.

[15] J. P. Halimi, B. Pradelle, A. Guermouche et al., "Reactive DVFS control for multicore processors," in *Proceedings of the IEEE Green Computing and Communications (GreenCom '13)*, pp. 102–109, August 2013.

[16] M. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '06)*, p. 14, Tampa, Fla, USA, November 2006.

[17] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron, "Strategies for energy-efficient resource management of hybrid programming models," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 144–157, 2013.

[18] L. Tan, Z. Chen, Z. Zong, D. Li, and R. Ge, "A2E: adaptively aggressive energy efficient DVFS scheduling for data intensive applications," in *Proceedings of the IEEE 32nd International Performance Computing and Communications Conference (IPCCC '13)*, pp. 1–10, IEEE, San Diego, Calif, USA, December 2013.

[19] L. Wang, G. Von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS," in *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '10)*, pp. 368–377, IEEE, Melbourne, Australia, May 2010.

[20] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi, "Emprical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster," in *Proceedings of the IEEE International Conference on Cluster Computing (Cluster '06)*, pp. 1–10, September 2006.

[21] S. U. Khan and I. Ahmad, "A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, pp. 346–360, 2009.

[22] M. Mezmaz, N. Melab, Y. Kessaci et al., "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud

computing systems," *Journal of Parallel and Distributed Computing*, vol. 71, no. 11, pp. 1497–1508, 2011.

[23] M. A. Awan and S. M. Petters, "Energy-aware partitioning of tasks onto a heterogeneous multi-core platform," in *Proceedings of the IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS '13)*, pp. 205–214, Philadelphia, Pa, USA, April 2013.

[24] J.-J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS '09)*, pp. 1–12, IEEE, Rome, Italy, May 2009.

[25] H.-R. Hsu, J.-J. Chen, and T.-W. Kuo, "Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, pp. 1061–1066, European Design and Automation Association, Munich, Germany, March 2006.

[26] W. Y. Lee, "Energy-saving DVFS scheduling of multiple periodic real-time tasks on multi-core processors," in *Proceedings of the 13th IEEE/ACM Symposium on Distributed Simulation and Real-Time Applications (DS-RT '09)*, pp. 216–223, IEEE Computer Society, October 2009.

[27] J. Luo and N. K. Jha, "Power-efficient scheduling for heterogeneous distributed real-time embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1161–1170, 2007.

[28] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Understanding the future of energy-performance trade-off via DVFS in HPC environments," *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp. 579–590, 2012.

[29] L. Tan, Z. Chen, Z. Zong, R. Ge, and D. Li, "A2E: adaptively aggressive energy efficient DVFS scheduling for data intensive applications," in *Proceedings of the IEEE 32nd International Performance Computing and Communications Conference (IPCCC '13)*, pp. 1–10, San Diego, Calif, USA, December 2013.

[30] M. A. Blackburn, *Five Ways to Reduce Data Center Server Power Consumption*, Green Grid, 2008.

[31] E. N. (Mootaz) Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Power-Aware Computer Systems*, B. Falsafi and T. N. Vijaykumar, Eds., vol. 2325 of *Lecture Notes in Computer Science*, pp. 179–197, Springer, Berlin, Germany, 2003.

[32] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*, pp. 13–23, ACM, June 2007.

[33] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *Proceedings of the ACM 11th International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '09)*, vol. 37, pp. 157–168, Seattle, Wash, USA, June 2009.

[34] T. Mudge, "Power: a first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.

[35] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[36] J. Hromkovič, *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*, Springer Science & Business Media, 2013.

[37] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[38] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, vol. 2, pp. 215–230, Ottawa, Canada, 2006.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 5706790, 14 pages
http://dx.doi.org/10.1155/2016/5706790

Research Article

Biobjective VoIP Service Management in Cloud Infrastructure

Jorge M. Cortés-Mendoza,[1] Andrei Tchernykh,[1] Fermin A. Armenta-Cano,[1] Pascal Bouvry,[2] Alexander Yu. Drozdov,[3] and Loic Didelot[4]

[1] *CICESE Research Center, Ensenada, BC, Mexico*
[2] *University of Luxembourg, Luxembourg City, Luxembourg*
[3] *Moscow Institute of Physics and Technology, Moscow, Russia*
[4] *MIXvoip S.A., Steinsel, Luxembourg*

Correspondence should be addressed to Andrei Tchernykh; chernykh@cicese.mx

Received 22 January 2016; Revised 3 July 2016; Accepted 16 August 2016

Academic Editor: Ligang He

Voice over Internet Protocol (VoIP) allows communication of voice and/or data over the internet in less expensive and reliable manner than traditional ISDN systems. This solution typically allows flexible interconnection between organization and companies on any domains. Cloud VoIP solutions can offer even cheaper and scalable service when virtualized telephone infrastructure is used in the most efficient way. Scheduling and load balancing algorithms are fundamental parts of this approach. Unfortunately, VoIP scheduling techniques do not take into account uncertainty in dynamic and unpredictable cloud environments. In this paper, we formulate the problem of scheduling of VoIP services in distributed cloud environments and propose a new model for biobjective optimization. We consider the special case of the on-line nonclairvoyant dynamic bin-packing problem and discuss solutions for provider cost and quality of service optimization. We propose twenty call allocation strategies and evaluate their performance by comprehensive simulation analysis on real workload considering six months of the MIXvoip company service.

1. Introduction

Voice over Internet Protocol (VoIP) has now become the most popular technology to communicate for long distance calling and is adopted all over the world. Together with general aspects of quality of service (QoS) of the Internet and other networks, like transmission rates, error rates, and other characteristics, VoIP adds new requirements: voice quality, service response time, throughput, loss, interrupts, jitter, latency, resource utilization, and so on. Hypervisor-level scheduling, traffic control, dynamic resource provisioning, and so forth are issues to address for the VoIP providers to ensure QoS and successful end-to-end business solution.

Effective VoIP scheduling involves many important issues: load estimation and prediction, performance analysis, system stability, call resource requirements estimation, routing, bandwidth limitation, resource selection for call allocation, and so forth [1–3].

Businesses provided VoIP systems are always looking for a way to cut down costs. Beloglazov et al. 2012 [4] consider efficiency of resource management deployed on the infrastructure and applications running on the system. One of the ways to reduce a cost is to avoid provisioning of more resources than required by users and QoS.

Cloud VoIP (CVoIP) solutions can offer even cheaper and scalable service by using virtualized telephone infrastructure in the efficient way. However, Tchernykh et al., 2015 [5], show that virtualization in cloud computing adds other complexity dimensions to the problem in terms of parameter variation, system uncertainty, dynamic consolidation of the virtual machines (VMs), and their migrations.

In this paper, we continue study presented by Cortés-Mendoza et al., 2015 [3], where we introduce a VoIP optimization model and study five call allocation strategies. We describe and analyze a model for cloud VoIP services focusing on two important aspects: QoS and VM utilization. We

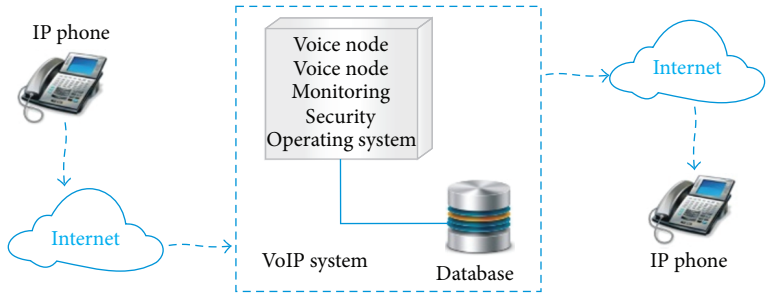


Figure 1: VoIP architecture.

take into account two main beneficiaries of the optimization technology: VoIP provider that runs its software on the cloud and end users.

While voice quality in real VoIPs is often seriously affected by the signaling overhead, end-to-end delay, jitter, packet loss, compression technique, hypervisor-level scheduling, and so forth, we restrict ourselves to voice quality affected by CPU usage during call processing. We believe that the focus in our model is reasonable and representative for real installations and applications.

In this paper, we provide a range of monoobjective and biobjective optimization solutions considering billing hours for VM running in a cloud and voice quality. We conduct the comprehensive simulation on real data and show that our scheduling strategies can provide a good compromise between saving money and voice quality. The paper makes the following contributions:

(i) We propose a set of on-line dynamic nonclairvoyant scheduling strategies to deal with VoIP calls in cloud environments. These strategies cover a wide domain of the VoIP biobjective problem, so that VoIP providers can select specific strategies depending on the goals.

(ii) We propose a novel function to ensure the VoIP QoS. This function considers the CPU utilization as a mean to evaluate the voice quality reduction.

(iii) We validate twenty strategies and evaluate their performance by comprehensive simulation analysis on real workload of the MIXvoip provider [6].

The paper is structured as follows. The next section briefly discusses VoIP service considering underlined infrastructure and software. Section 3 reviews related works. Section 4 presents several factors that have an impact on the QoS and provider cost. Section 5 provides the problem definition and corresponding model. Section 6 describes methodology of the analysis. Section 7 describes approaches for VoIP call allocation and corresponding algorithms. Section 8 describes our experimental setup, workload, and studied scenarios. Section 9 presents experimental analysis of the provider cost when quality of service is guaranteed. Section 10 analyzes a general biobjective problem. Section 11 concludes the paper by presenting the main contribution of the work and future research directions.

2. Internet Telephony

The Internet telephony VoIP refers to the provisioning of voice communication services over the Internet rather than via the traditional telephone ISDN network (ISDN (Integrated Services Digital Network) is a set of standards for digital transmission over ordinary telephone copper wire). One of the reasons of its wide acceptance is significant reduction of calling rates.

The scalability requires the service availability all the time for any number of users. To deal with increasing number of clients, providers may invest in a large infrastructure to avoid loss of calls (hence, users). In the case of overprovisioning, the infrastructure is underutilized most of the time.

The clients connect to a voice server, which is the main part of the VoIP telephony system (Figure 1). The voice nodes communicate with the database in the system, where all the users are registered, and calls are recorded with details such as destination and duration. They provide software to emulate a telephone exchange, gateways, interconnection switches, session controllers, firewall, and so forth.

The voice nodes handle calls with different features such as voicemail, call forwarding, music on hold, and conference calls depending on customers. They provide signaling, voice signal digitization, encoding, and so forth. In order to use VoIP services, an Internet connection and an IP hard-phone or soft-phone are needed.

Traditional VoIP solutions are not scalable. Drawbacks arise when the hardware reaches its maximum capacity. To scale it, it is necessary to increase or replace existing hardware. Overprovisioning and, hence, cost overrunning are not an efficient solution even with the growing number of the customers and potential safety of being able to deliver services during peak hours or abnormal system behavior.

A CVoIP can further reduce costs, add new features and capabilities, provide easier implementations, and integrate services that are dynamically scalable. Other benefits include data transfer availability, integrity, and security.

Cloud-based VoIP solutions allow reducing an importance of a Build-To-Peak approach. The virtual infrastructure can be easily scalable.

In this solution, the voice nodes are operated by VMs. Distributed cloud-based VoIP architecture assumes that voice nodes are distributed geographically; hence, they are grouped in different locations (data centers). To deploy and effectively

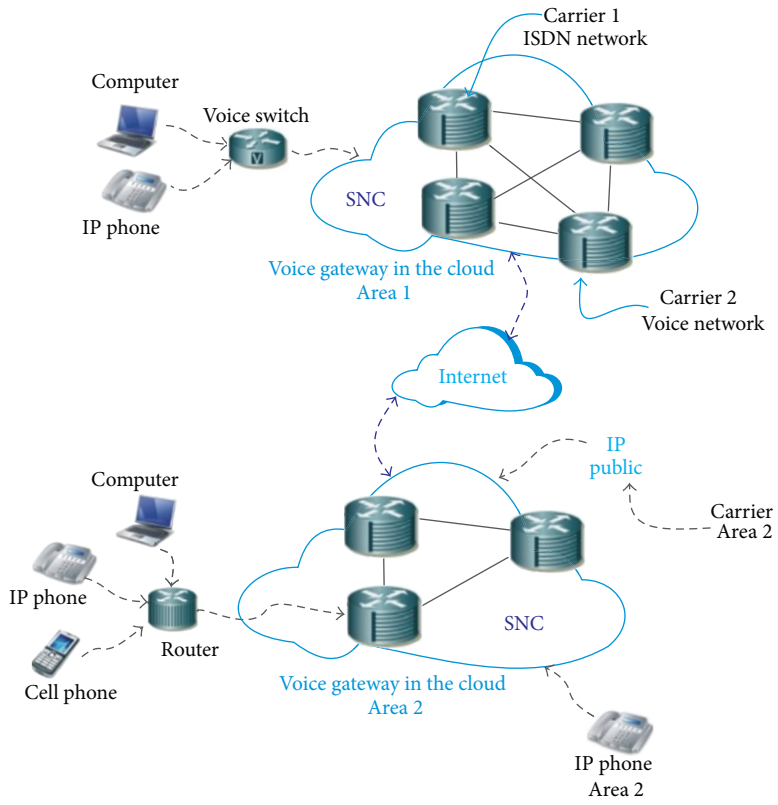


Figure 2: SNC deployment [3].

manage telephones via clouds, different characteristics need to be improved. The most important is the utilization of the virtual infrastructure and voice quality.

The advantage of cloud-based solution is in increased scalability and low cost. However, it has several unsolved problems. To optimize the overall system performance and reduce provider cost, the VM utilization has to be high, but it reduces quality of the calls (see Section 4). Hence, load of the VoIP servers should be reduced to guarantee the QoS. On the other hand, the idle time increases the useless expenses of the VoIP provider.

The most important cause of the load imbalance is the dynamic nature of the problem and system. The objective is to maximize VoIP system performance by minimizing the number of processing units without overloading them. It can improve the provider cost and guarantee QoS.

2.1. Infrastructure. MIXvoip company [6] presents telephony combining cloud service with smart business telephony, VoIP, and other telephony services.

It developed the concept of the super node (SN) and super nodes cluster (SNC) to enrichment features for telephone exchanges (Figure 2).

SNC is a set of SNs deployed in a cloud and interconnected logically at a local level. It provides short path between two local users. This deployment brings redundancy on a given geographical area but ensures a high voice quality between the SNC nodes through the public Internet.

As shown in Figure 2, a user call is allocated to the nearest SN in his area. This deployment allows providing services near ISDN quality in a public IP network.

2.2. Software. Asterisk is the most known Private Branch Exchange (PBX) software that includes all of the components necessary to build scalable phone systems (see Madsen et al. 2011) [10]. It allows making and processing calls and connecting to other telephone services, such as the public switched telephone network (PSTN) and VoIP services. It is a framework for building multiprotocol, real-time communication solutions providing a powerful control over call activity.

Delivering information and transferring data are based on protocols, such as Session Initiation Protocol (SIP) and Real Time Protocol (RTP). SIP is the protocol for signaling, establishing presence, locating users, setting, modifying, and tearing down sessions between end-devices. It is used for controlling communication sessions such as voice and video calls over IP networks. The media transportation is provided via RTP. Codecs are used for converting the voice portion of a call in audio packets to transmit over RTP streams.

The VoIP system consists of multiple heterogeneous voice nodes that run and handle calls. Each node has Asterisk running processes. Each Asterisk instance has a unique IP address that is used by end users to connect inside and outside the network.

3. Related Work

Different techniques have been introduced in recent years in order to overcome the challenges of VoIP. However, VoIP scheduling for QoS and provider cost optimization are still insufficiently studied.

So, [11] (2011) analyzes dynamic scheduling and persistent scheduling for VoIP services in wireless orthogonal frequency division multiple access systems. The author develops analytical and simulation models to evaluate the performance of VoIP services in terms of the average throughput and the signaling overhead according to the scheduling schemes.

Lee et al. (2006) [12] analyze the performance of three scheduling algorithms (Unsolicited Grant Service (UGS), real-time Polling Service (rtPS), and extended real-time Polling Service (ertPS)) for IEEE 802.16e standard covering mobile broadband wireless systems. The authors use simulation to show that ertPS algorithm with Enhanced Variable Rate Codec (EVRC) and silence suppression can support more calls compared with the UGS and rtPS algorithms, on 21% and 35%, respectively.

Folke et al. (2007) [13] analyze four scheduling algorithms: Proportional Fair (PF), Maximum Rate (MR), MR with the minimum bit rate (MR_{min}), and MR with strict delay (MR_{delay}) for mix of conversational (VoIP) traffic and interactive (web) traffic. All strategies are tested with varying scheduling delay budgets and loads. The authors show that a scheduler that gradually increases the VoIP priority and considers the user's current possible rate performs well. However, a more drastic increase in VoIP priority is needed when the delay budget is short. Furthermore, attempting to uphold quality for both VoIP and web traffic makes the system sensitive to overload situations.

Bayer et al. (2010) [14] analyze on demand scheduling, uncoordinated resource coordination scheme, coordinated resource coordination scheme, and VoIP aware resource coordination scheme for TDMA based access control in mesh networks in the IEEE 802.16 standard. The authors show that the mesh networks are able to support VoIP with good quality when a persistence scheduling is applied. Compared to other resource coordination schemes the VoIP aware scheduler significantly increases the number of supported calls.

Wu et al. (2014) [15] present a real-time scheduling framework in virtualized environment that considers real-time constraints of applications. The authors propose a mechanism called multicore dynamic partitioning to divide physical CPUs into two pools dynamically according to the scheduling parameters of VMs. They use global earliest deadline first strategy to schedule calls on VMs, with an Asterisk instance, to improve CPU usage and guarantee the call quality.

Mazalek et al. (2015) [16] study the impact of the IPSec encryption on the CPU utilization, bandwidth, and voice quality. The authors show a significant effect of voice payload period on the CPU utilization and bandwidth. They save up to 40–60% of bandwidth when the period is chosen properly. The call quality, expressed by mean opinion score (MOS) scale, remains almost constant up to the moment, when the CPU utilization is close to 80%.

Table 1: Processor utilization for 1 call without transcoding [7].

Protocol	Codec	10 calls	1 call
SIP/RTP	G.711	2.36%	0.236%
SIP/RTP	G.726	2.13%	0.213%
SIP/RTP	GSM	2.58%	0.258%
SIP/RTP	LPC10	1.92%	0.192%

Costa et al. (2015) [17] show that Asterisk PBX server is able to provide VoIP communication capabilities with an acceptable MOS quality. The authors use the blocking probability metric to measure the capacity of the VoIP server and MOS to assess the quality of the voice calls. The experimental results show that the Asterisk PBX using SIP effectively handled more than 160 concurrent voice calls with a blocking probability below 5%.

Cheng et al. (2015) [18] present and compare the SRT-Xen scheduler with other four schedulers (Credit, Credit2, rtglobal, and rtpartition). They focus on real-time-friendly scheduling to improve the management of the virtual CPUs' queueing. They use an instance of Asterisk to evaluate the performance of the strategies and speech quality. SRT-Xen achieves at least 13.61% more sessions with perceptual evaluation of speech quality >4.

4. VoIP Quality of Service

4.1. Utilization. Calls have different impact on the processor utilization depending on the operations performed by Asterisk, when the calls are being established. If transcoding operations are performed, the utilization is higher than that when transcoding is not used. In the latter case, Asterisk is in charge of only routing the call. However, depending on the codec, the processor load is influenced as well. Table 1 shows processor utilization for call without transcoding presented by Montoro and Casilari (2009) [7].

VoIP gateways support a larger number of codecs and DSP modules (Digital Signal Processing): G.711, GSM, LPC10, Speex. G.711 A-law and U-law PCM, G.726 ADPCM, G.728 LD-CELP, G.729 CS-ACELP, G.729a CS-ACELP, G.729 Annex-B, G.729a Annex-B, G.723.1 MP-MLQ, G.723.1 ACELP, G.723.1 Annex-A MP-MLQ, G.723.1 Annex-A ACELP, and so forth. Some codec compression techniques require more processing power than others. Examples of the compression method are presented by Cisco [9]:

PCM: Pulse Code Modulation

ADPCM: Adaptive Differential Pulse Code Modulation

LDCELP: Low-Delay Code Excited Linear Prediction

ACELP: Algebraic-Code-Excited Linear Prediction

MP-MLQ: Multi-Pulse, Multi-Level Quantization

CS-ACELP: Conjugate-Structure Algebraic-Code-Excited Linear Prediction.

In [8], the authors present results of the benchmark test that includes stress testing of queue calls, VoIP calls,

Table 2: Queue calls [8].

Activity test	Jitters	CPU usage	Simultaneous calls	CPU usage per 1 call
5 calls to queue	None	14%	10	1.40%
10 calls to queue	None	18%	20	0.90%
15 calls to queue	None	28%	30	0.93%
20 calls to queue	None	36%	40	0.90%
30 calls to queue	None	67%	60	1.11%
40 calls to queue	None	84%	80	1.05%

Table 3: MOS score [9].

Compression	Bit rate (kbps)	MOS score	Compression delay (ms)
G.711 PCM	64	4.1	0.75
G.726 ADPCM	32	3.85	1
G.728 LD-CELP	16	3.61	3 to 5
G.729 CS-ACELP	8	3.92	10
G.729 × 2 encodings	8	3.27	10
G.729 × 3 encodings	8	2.68	10
G.729a CS-ACELP	8	3.7	10
G.723.1 MP-MLQ	6.3	3.9	30
G.723.1 ACELP	5.3	3.65	30

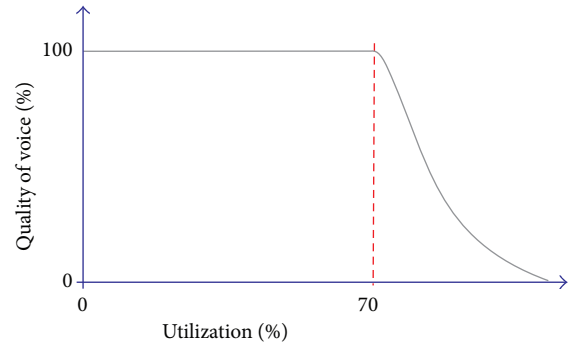


Figure 3: Voice quality versus processor load (utilization).

and extension to extension calls. Queuing calls are used by call centers that prefer to answer to the incoming calls automatically and place them in a queue instead of rejecting them. Queuing allows the acceptance of more calls into the system than existing extensions or agents capable of answering them. While on hold, the callers receive different announcements (position in the queue) followed by music.

Considering Tables 1 and 2, we conclude that CPU can process from 70 to 500 calls with 100% of utilization.

4.2. Quality of Service. The VoIP QoS is determined by the quality of voice, transit time of packets across the Internet, queuing delays at the routers, packet travel time from source to destination, jitter as deviations of the packet interarrivals, packet loss, call setup and tear downtime, and so forth.

The quality of voice is a subjective response of the listener. A common benchmark used to determine the quality of sound produced by specific codecs is the Mean Opinion Score (MOS). Listeners judge the quality of a voice sample that corresponds to a particular codec on a scale of 1 (bad) to 5 (excellent). The scores are averaged to provide the MOS for that sample. Table 3 shows the relationship between codecs and MOS scores presented by Cisco [9].

Cortés-Mendoza et al. (2015) propose using processor utilization in order to ensure QoS. Each codec provides a certain quality of voice only if processor utilization is low enough. Theoretically, processor utilization of 100% provides the best expected performance. However, in [8], the authors show that 20 calls via a VoIP provider produced no jitters; CPU usage in total was 19%. With increasing number of calls up to 90 and utilization up to 85%, CPU cannot be able to handle the stress anymore and jitters and broken audio symptoms will appear.

Considering different types of calls with different codecs, we use a threshold of 70% to ensure a high voice quality.

Figure 3 shows that the voice quality is reduced fast when the processor utilization exceed 70%.

4.3. VoIP Provider Costs. VoIP provider costs are primarily tied to their assets and the maintenance of these assets. For example, providers have an infrastructure that needs to be powered and cooled. It has storage arrays containing storage disks, and these arrays are connected to chassis which are all housed. So, major provider costs can be categorized as servers cost (computing, storage, software, and associated VoIP components), infrastructure cost (power distribution, cooling equipment, space for facilities, etc.), operational cost (energy, cooling, etc.), and network cost (links and transit equipment). A number of other costs exist.

To offer competitive prices to prospective customers VoIP providers should optimize the process. Inefficient resource management has a direct negative effect on performance and cost.

Virtualization technologies allow creating VoIP virtual servers, which can then be hosted in data centers and rented out (leased) on a subscription basis to any scale.

In a typical cloud scenario, a VoIP provider has the choice between different resources that are available on demand from cloud providers with certain service guarantees. These service levels are mainly distinguished by the amount of computing power guaranteed to receive within a requested time and a cost per unit of execution time the VoIP provider has to pay. This cost depends on the type of requested computing resources, for instance, VMs with different performance.

In order to evaluate the provider cost for cloud solution, we use a metric that is useful for systems with VM. It must allow the provider to measure the cost of the system in terms of number of demanded VMs and time of their using.

In this paper, two criteria are considered for the model: the billing hours for VMs to provide a service and their utilization to estimate quality of service.

In the first scenario, we consider single-objective optimization problem minimizing the total cost of VMs with given restrictions. In order to ensure good QoS, the utilization of the VMs is kept under the certain threshold (e.g., 70%).

In the second scenario, we consider the biobjective optimization approach that is not restricted to find a unique solution but a set of solutions known as a Pareto optimal set. In this case, we minimize two conflicting objectives: the cost of VMs and QoS degradation. A tradeoff between the two objectives depends on the VoIP provider's preference.

5. Model

We address the model for VoIP in distributed cloud environment with heterogeneity of resources with different number of servers, execution speed, amount of memory, bandwidth, and so forth.

Let us consider that VoIP cloud infrastructure consists of m heterogeneous super node clusters SNCs : $\mathrm{SNC}_1, \mathrm{SNC}_2, \ldots, \mathrm{SNC}_m$ with relative speeds $s_1, s_2, \ldots . s_m$. Each SNC_i, for all $i = 1, \ldots, m$, consists of m_i SNs. Each SN_k^i, for all $k = 1, \ldots, m_i$, runs $k_i(t)$ VM at time t. We assume that VMs of one SNC are identical and have the same processing capacity.

The SNC contains a set of routers and switches that transport traffic between the SNs and the outside world. A switch connects a redistribution point or computational nodes. The connections of the processors are static but their utilization is changed. The SNC interconnection network architecture is local. The interconnection between SNCs is provided through public Internet.

We consider n independent calls or jobs $J_1, J_2, \ldots, J_n$ that must be scheduled on set of SNCs. The job J_j is described by a tuple $\{r_j, p_j, u_j\}$ that consists of its release date $r_j \geq 0$, duration p_j (lifespan), and contribution to the processor utilization u_j. The release time of a job is not available before the job is submitted, and its duration is unknown until the job has been completed. The utilization is a constant for a given job that depends on the used codec and is normalized for the slowest machine.

In order to evaluate the system performance, we use metrics that are useful for VoIP systems, where traditional measures such as makespan, throughput, and response time become irrelevant.

For this kind of systems, the metrics must allow the provider to measure the performance of the system in terms of financial attraction which helps him to assure benefits as well as user satisfaction for the service.

Two criteria are considered in the analysis: Minimization of the service provider cost and minimization of the quality of service degradation.

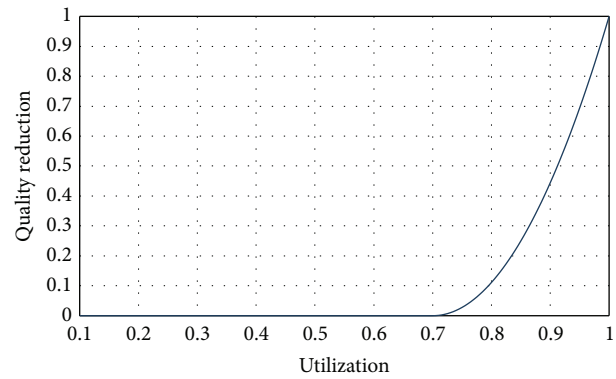


Figure 4: Voice quality reduction versus processor load (utilization).

We define the provider cost model by considering a function that depends on the number of VMs and their running time.

We denote the number of billing hours in SNC_i by $\overline{m}_i = \int_{t=0}^{C_{\max}} k_i(t) \cdot m_i dt$ and run in m SNC by $\overline{m} = \sum_{i=1}^{m} \overline{m}_i$. The VM is described by a tuple $\{\mathrm{vmu}_i(t)\}$, where $\mathrm{vmu}_i(t)$ is the utilization (load) of VM_i at time t. VM hosts Asterisk running process that handles calls.

As an optimization criterion, we introduce a quality reduction function based on the VMs utilization (Figure 4).

6. Methodology of Analysis

To derive bounds of the provider cost, we consider two scenarios. The maximum cost can be archived if provider guarantees the voice quality with quality reduction equal to 0 by setting the utilization threshold to 70%. Then, in the second scenario, we realize a biobjective analysis, where no threshold is used (100% of utilization is allowed), to study the compromise between the provider cost and voice quality reduction.

6.1. Degradation in Performance. To choose a good strategy, we perform an analysis based on the degradation methodology proposed in [19] and applied for scheduling in [20]. It shows how the metric generated by our algorithms gets closer to the best found solution.

The analysis is conducted as follows. First, we evaluate the degradation in performance (relative error) of each strategy relatively to the best performing strategy as follows:

$$(\gamma - 1) \cdot 100, \quad \text{with } \gamma = \frac{\text{strategy metric value}}{\text{best found metric value}}. \tag{1}$$

Then, we average these values for all scenarios and rank the strategies. The best strategy with the lowest average performance degradation has rank 1. Note that we try to identify strategies, which perform reliably well in different scenarios; that is, we try to find a compromise that considers all of our test cases. For example, the rank of the strategy could not be the same for any of the scenarios individually.

6.2. Biobjective Analysis. In multiobjective optimization, one solution can represent the best solution concerning provider cost, while another solution could be the best one concerning the QoS. The goal is to choose the most adequate solution and obtain a set of compromise solutions which represents a good approximation to the Pareto front.

Two important characteristics of a good solution technique are convergence to the Pareto front and diversity to sample the front as fully as possible. A solution is Pareto optimal if no other solution improves it in terms of all objective functions. Any solution not belonging to the front can be considered of inferior quality to those that are included. The selection between the solutions included in the Pareto front depends on the system preference. If one objective is considered more important than the other one, then preference is given to those solutions that are near-optimal in the preferred objective, even if values of the secondary objective are not among the best obtained.

Often, results from multiobjective problems are compared via visual observation of the solution space. One of formal and statistical approaches uses a set coverage metric $\mathrm{SC}(A, B)$ that calculates the proportion of solutions in B, which are dominated by solutions in A:

$$\mathrm{SC}(A, B) = \frac{\{|b \in B|\,; \exists a \in A; a \leq b\}}{|B|}. \quad (2)$$

A metric value $\mathrm{SC}(A, B) = 1$ means that all solutions of B are dominated by A, whereas $\mathrm{SC}(A, B) = 0$ means that no member of B is dominated by A. This way, the larger the value of $\mathrm{SC}(A, B)$, the better the Pareto front A with respect to B. Since the dominance operator is not symmetric, $\mathrm{SC}(A, B)$ is not necessarily equal to $1 - \mathrm{SC}(A, B)$, and both $\mathrm{SC}(A, B)$ and $\mathrm{SC}(B, A)$ have to be computed for understanding how many solutions of A are covered by B and vice versa.

7. Call Allocation

In our model, CPU utilization is a key performance metric for VoIP quality of service measurement. It can be used to track QoS reductions, when it increases above the certain threshold, or improvement, when it is below, and it is useful for VoIP QoS problem studying.

The concept of VM utilization used in our study is simple. Assume that the VM is allocated on a single core processor of 2.0 GHz. VM utilization in this scenario is the percentage of time the processor spends doing VM work (as opposed to being idle). If the processor does 1 billion cycles worth of VM work in a second, it is 50% utilized for that second.

In general, monitoring CPU utilization, where VM is running, is straightforward: from a single percentage of CPU utilization to the more in-depth statistics. We can also gain a bit of insight into how the CPU is being used. To gain more detailed knowledge regarding VM utilization, we must examine all details of the VM parameters, software installed, and hardware of a system.

There are a lot of factors that contribute to the processor utilization. In our case, we reduce ourselves to consider Asterisk running processes and calls.

The call allocation problem is similar to a well-known one-dimensional on-line bin-packing problem, the classical NP-hard optimization problem with high theoretical relevance and practical importance. Bin-packing concerns placing items of arbitrary height into a one-dimensional space (bins with fixed capacity) efficiently.

Bin-packing remains one of the classical difficult problems. Scientists have analyzed and studied this computational puzzle for decades, yet none have obtained an algorithm which derives the optimal solution in reasonable amount of time. We consider an on-line variant of the problem in which items are received one by one.

Bins represent VMs, and the items height defines the call contribution to the processor utilization. Before info about the next call is revealed, the scheduler needs to decide whether the call is packed in the currently available VMs or a new VM must be rented. The scheduler only knows the contribution of the call to the processor utilization u_j due to the used codec. All decisions have to be made without knowledge of duration of the call, call arrival rate, and so forth.

The principal novelty of this problem variation lies in the temporal existence of the items. After a call lifespan is reached, the VMs can free space for processing more calls, so the state of the VMs is determined not only by the decision maker during call allocations. Unlike the standard formulation, bins are always open and dynamic and even completely packed. Items in bins can be terminated (call termination) and utilization can be changed at any moment.

As mentioned in Section 6, we consider two scenarios. In the first scenario, the bin size is equal to 0.7 which corresponds to 70% of VM utilization, so that the quality reduction is zero. In the second scenario, the bin size is equal to 1 which corresponds to 100% of VM utilization, so that the quality reduction can appear.

On both scenarios, we do not sort the input items due to the fact that we face an on-line bin-packing problem. Instead, we can sort bins based on their utilization.

We study twenty strategies (Table 4), Rand, RR, FFit, Bfit, WFit, MaxFTFit, MidFTFit, MinFTFit, RR_05, RR_10, RR_15, Wfit_05, Wfit_10, Wfit_15, BFit_05, BFit_10, BFit_15, FFit_05, FFit_10, and FFit_15, and evaluate their performance with the real workload considering six months of the MIXvoip company service.

We categorize all strategies in four groups by the type and amount of information used for allocation decision (1) knowledge-free (KF), with no information about applications and resources; (2) utilization-awareness (UA) with CPU utilization information; (3) time-awareness (TA) with VM rental time information; and (4) time-awareness with CPU utilization information (TA + UA).

In our previous work, Cortés-Mendoza et al. (2015) [3] study three well-known bin-packing strategies adapted for the described problem, First-Fit (FFit), Best-Fit (Bfit), and Worst-Fit (Wfit), and two commonly used allocation strategies, Round Robin (RR) and Random (Rand).

The significant contribution of this paper compared with the previous work is that we analyze twenty strategies, consider different scenarios solving monoobjective and biobjective problems, and provide a deeper study of our

Table 4: Call allocation strategies.

	Description	
KF	Rand	Allocates job j to VM randomly using a uniform distribution
	RR	Allocates job j to VM using a Round Robin algorithm
UA	FFit	Allocates job j to the first VM able to execute it
	BFit	Allocates job j to VM with smallest utilization left
	WFit	Allocates job j to VM with largest utilization left
TA	MaxFTFit	Allocates job j to VM with farthest finish time
	MidFTFit	Allocates job j to VM with finish time between farthest and closest
	MinFTFit	Allocates job j to VM with closest finish time
	RR_05	Allocates job j to VM that finishes not less than in 5, 10, and 15 minutes using the RR strategy
	RR_10	
	RR_15	
TA + UA	BFit_05	Allocates job j to VM that finishes not less than in 5, 10, and 15 minutes using the WFit, BFit, and FFit strategies
	BFit_10	
	BFit_15	
	FFit_05	
	FFit_10	
	FFit_15	
	WFit_05	
	WFit_10	
	WFit_15	

```
Input: Voice node list (VNlist) and call.
Output: Allocation of call in one voice node.
(1)  Sort VNlist by utilization on decreasing order.
(2)  assigned ← false
(3)  node_index ← 1
(4)  Do
(5)      node_voice ← get(VNlist, node_index)
(6)      Add call to node_voice
(7)      if utilization of node_voice <= 0.7 then
(8)          assigned ← true
(9)      else
(10)         remove call from node_voice
(11)         node_index ← node_index + 1
(12)     endif
(13) While (size of VNlist >= node_index and
(14)     assigned = false)
(15) If assigned = false then
(16)     Create new_node_voice
(17)     Add call to new_node_voice
(18)     Insert new_node_voice into VNlist
(19) Endif
```

Algorithm 1: Best fit (BFit).

```
Input: Voice node list (VNlist), time and threshold.
Output: A voice node list for processing call.
(1) Create new_VNlist
(2) For each node_voices on VNlist
(3)     If time_end(node_voice) <= time + threshold
(4)         Add node_voice to new_VNlist
(5) endfor
(6) return new_VNlist
```

Algorithm 2: Admissible VMs list (AVML).

algorithms performance taking into account billing hours and quality reduction.

Algorithm 1 describes the BFit strategy, where voice nodes in the list are sorted in decreasing order of their utilization. We use the term the voice node instead of VM to have coherence with the call allocation terminology.

Line (1) may be changed depending on the strategy of allocation. For example, the list is sorted in increasing order in WFit strategy, and this line is not used in FFit strategy [21].

The main idea of the time-aware approach is to allocate calls to VM taking into account the finishing time of rented hours.

The goal is to allocate calls to VM to reduce number of billing hours. We try to avoid next hour renting due to continuation of the call over the rented hour.

For instance, MaxFTFit schedules the call to VM with farthest away finish time. MinFTFit schedules the calls to the voice node with nearest finish time. It tries to use already running voice nodes. The objective of MidFTFit is not to allocate calls to VMs that are not in beginning or end of the rental time.

We also introduce the time-aware versions of RR and WFit strategies (RR_05, RR_10, RR_15, WFit_05, and WFit_10), where we do not allocate calls to VMs in which rented time is finished in certain threshold. By these thresholds, we try to avoid next hour renting due to continuation of this call over the rented hour. It could reduce the number of billing hours. We study three thresholds: 5, 10, and 15 minutes before the end of renting hour.

For these strategies, the algorithm has a new procedure, named AVML (Admissible VMs List) (see Algorithm 2).

8. Experimental Setup

8.1. Simulation Toolkit. All experiments are performed using the CloudSim [22], a framework for modeling and simulation of cloud computing infrastructures and services. It is a standard trace based simulator that is used to study cloud resource management problems. We have extended CloudSim to include our algorithms for call allocation, supporting dynamic calls arrival, updating the system parameters before scheduling decisions, using the utilization of the resources, dynamically creating VMs, and providing statistical analysis using the java (JDK 7u51) programming language.

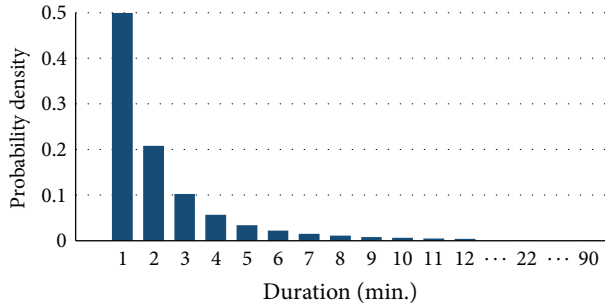


Figure 5: Call duration distribution.

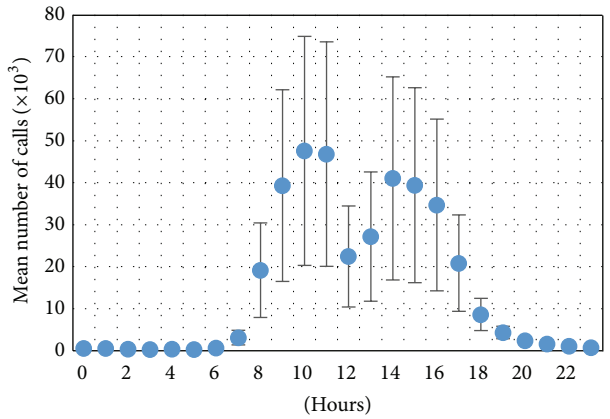


Figure 6: Mean number of calls per hour in a day during 24 weeks with the standard deviation.

Parameters are directly taken from traces of real VoIP service studied by Simionovici et al. (2015) [2]. We use SWF (Standard Workload Format) with four additional fields to process the calls.

8.2. Workload. The workload is a set of registered phone calls that have been registered by the VoIP system. It is recorded in the Call-Detail-Record (CDR) database with the following information: index of the call, ID of the user who makes the call, IP of the phone where the call is placed from, IP of the local phone, destination of the call, destination country code, destination country name, telecommunications service provider, beginning of the call (timestamp), duration of the call (in seconds), duration of a paid call, cost per minute; and so forth.

Supported call statistics could include incoming/outgoing call attempts, whether successful or not, calls rejected or failed, number of calls whose connected time is less than the configured minimum call duration (MCD), number of calls losing more than the configured number of packets, number of calls encountering more than the configured amount of latency and jitter, calls disconnected, and so forth.

Total call distributions per hour and per day during six months (from 1 November 2014 to 17 April 2015) are presented in Cortés-Mendoza et al. (2015) [3].

They demonstrate typical behavior for business customers: two peak hours, 8–11 AM and 13–17 PM. Over a week, the traffic is high from Monday till Friday, while for weekends it decreases considerably.

Figure 5 shows the call duration distribution during the six months, which depends significantly on the clients (e.g., call centers, schools, and business companies). In our example, the duration of the majority of the calls is short (e.g., 1–5 minutes).

Dang et al. (2004) [23] showed that the call arrival process is fitted by a Poisson process and the call duration distribution by a generalized Pareto distribution with parameter values indicating finite variances. The authors tested a series of probability distributions and showed that the model agrees well with the data in high-density regions and also fits the low-density regions, known as tails of the distribution (Figure 5).

For the analysis, we use 24 workloads; each includes phone calls made during one week. Figure 6 shows mean number of calls per hour in a day during 24 weeks.

During the weekends, the workload is low. It needs only one VM to process it. For this study, we removed the jobs of the weekends because they can be replaced with 24 billing hours per day. One VM is always running even with no calls.

8.3. Scenarios. In our scenarios, each VM runs an instance of Asterisk (voice node). The VMs are deployed by several super node voices (SNs) and all of them belong to one SNC. The VoIP providers rent the VMs by hours [24]; when the VM rental time is finished the VM can be turned off only if VM is not processing any calls; in other cases, this VM continues running for one more hour.

9. Scenario 1: Cost Analysis with Guaranteed Quality of Service

In the first scenario, we evaluate the provider cost generated by the twenty strategies: BFit, BFit_05, BFit_10, BFit_15, FFit, FFit_05, FFit_10, FFit_15, MaxFTFit, MidFTFit, MinFTFit, Rand, RR, RR_05, RR_10, RR_15, WFit, WFit_05, WFit_10, and WFit_15.

We use the utilization threshold as the constraint to guarantee the quality of service.

Figure 7 displays the number of billing hours during 24 weeks. We see that the workload is low during weeks 8 and 9, so that the difference of billing hours generated by strategies is about 50. In other weeks, the dispersion is higher up to 160 billing hours in week 5.

Table 5 shows the average number of billing hours during considered 24 weeks. BFit and FFit are shown to be the best strategies using 252.08 and 252.42 billing hours per week on average to deal with given workload. WFit and MinFTFit are worst ones with 351.08 and 363.96 billing hours, respectively.

RR_05, RR_10, and RR_15 strategies have a better performance than non-time-aware RR. Similarly, WFit_05, WFit_10, and WFit_15 are better than WFit. The difference between the best strategy, BFit, and worst one, MinFTFit, is about 111 billing hours per week on average.

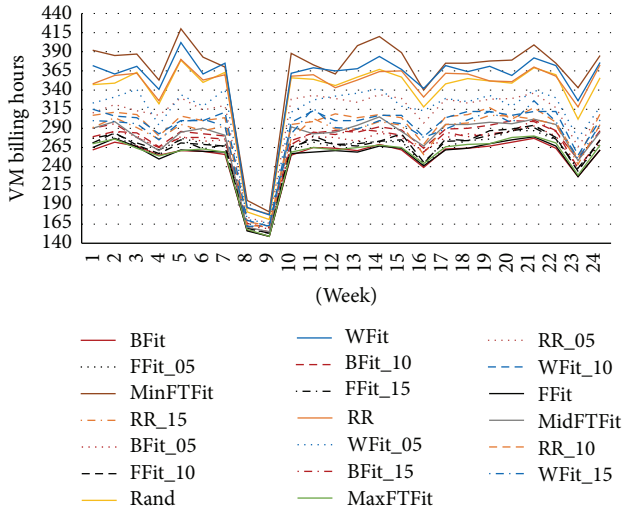


Figure 7: The number of billing hours during 24 weeks.

10. Scenario 2: Biobjective Analysis

10.1. Degradation. In multiobjective analysis, the problem can be simplified to a single objective problem through different methods of objective weighted aggregation. There are various ways to model preferences; for instance, they can be given explicitly to specify the importance of every criterion or a relative importance between criteria. This can be done by a definition of criteria weights or criteria ranked by their importance.

In this section, we perform a joint analysis of two metrics according to the mean degradation methodology described in Section 6.1.

First, we present the analysis of the billing hours for rented VMs and quality reduction separately. Then, we find the strategy that generates the best compromise between them.

In Table 6, we present the average degradation of billing hours, quality reduction, and their means. The last three columns of the table contain the ranking of each strategy regarding the provider cost, quality, and their means. Rank-BH is based on the billing hours' degradation. Rank-QR refers to the position in relation to the degradation of quality reduction. Rank is the position based on the averaging two rankings.

We see that the best strategy for the cost optimization is BFit which allocates calls based on best fit strategy, where we put the call into the fullest VM, which leaves the least utilization left over. However, it is the worst strategy for the voice quality. It tends to increase utilization and reduce quality.

The best strategy for the voice quality is WFit, where we put the call into the VM, which leaves most of utilization left over. It tends to underutilize VMs keeping the quality but increases VM number and renting cost.

A good compromise is MaxFTFit strategy that allocates the call to VM that finishes his hour far away.

Table 5: Average weekly billing hours.

Rank	Strategy	VM billing hours
1	BFit	252.08
2	FFit	252.42
3	MaxFTFit	254.71
4	FFit_05	259.46
5	FFit_15	261.75
6	FFit_10	261.79
7	BFit_05	266.13
8	BFit_15	269.21
9	BFit_10	273.25
10	MidFTFit	276.08
11	RR_15	279.79
12	WFit_15	283.79
13	RR_10	289.00
14	WFit_10	290.42
15	RR_05	306.88
16	WFit_05	311.29
17	Rand	336.29
18	RR	340.46
19	WFit	351.08
20	MinFTFit	363.96

10.2. Solution Space and Pareto Fronts. To solve the general biobjective problem, we want to obtain a set of compromise solutions that represent a good approximation to the Pareto front. This is not formally the Pareto front as an exhaustive search of all possible solutions is not carried out but rather serves as a practical approximation of a Pareto front.

Figure 8 shows the solution sets for the twenty strategies obtained based on 109 days of workload. This two-dimensional solution space represents a feasible set of solutions that satisfy the problem constraints. Note that we address the problem of minimizing cost and maximizing the quality. For better representation, we convert it to the minimization of two criteria: degradations of both the cost and quality reduction.

The solution space covers a range of values of cost degradation from 0 to 0.65, whereas values of quality reduction degradation are in the range from 0 to 0.26.

We see that the solution space is divided in three groups located in right lower side, left lower side, and in the middle. BFit, FFit, and MaxFTFit are located in the lower right side being among the best solutions in terms of the billing hours. They outperform other strategies, like RR, which are in current use for VoIP service. WFit is located in the left side being among the best solutions in terms of quality reduction. The three versions of time-aware WFit (WFit_05, WFit_10, and WFit_15) have a good behavior.

WFit is the best for quality reduction degradation (QRD $=$ 0). The range of the cost degradations is from 0.16 to 0.56. WFit_05 increases the QRD to 0.017 but reduces the cost up to 0.05. For WFit_10, the QRD increases from 0.017 to 0.06, but the cost reduces to 0.023.

Table 6: Degradation and ranking.

Strategy	Billing hours (BH)	Quality reduction (QR)	Mean	Rank BH	Rank QR	Rank
BFit	0.263	21.099	10.681	1	20	4
BFit_05	6.911	17.930	12.420	7	16	6
BFit_10	8.405	17.342	12.874	9	13	5
BFit_15	8.091	17.240	12.665	8	12	3
FFit	0.535	21.031	10.783	2	19	4
FFit_05	3.483	18.758	11.120	4	18	5
FFit_10	4.638	18.069	11.354	5	17	5
FFit_15	4.812	17.819	11.315	6	14	3
MaxFTFit	2.096	17.835	9.965	3	15	1
MidFTFit	10.536	16.442	13.489	10	11	4
MinFTFit	39.147	12.800	25.973	20	10	7
Rand	31.263	1.094	16.178	17	4	4
RR	32.681	0.732	16.707	18	2	3
RR_05	22.542	2.144	12.343	15	5	3
RR_10	15.388	4.666	10.027	13	7	3
RR_15	11.772	6.980	9.376	11	9	3
WFit	35.435	0.000	17.718	19	1	3
WFit_05	23.904	1.085	12.494	16	3	2
WFit_10	16.606	3.301	9.954	14	6	3
WFit_15	13.292	5.428	9.360	12	8	3

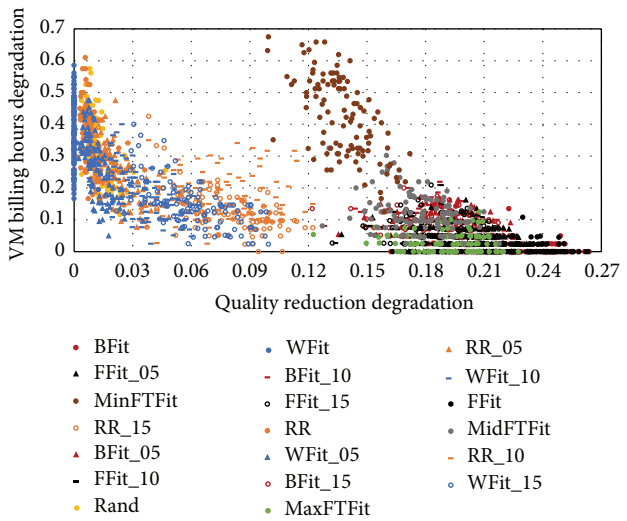


Figure 8: The solution space.

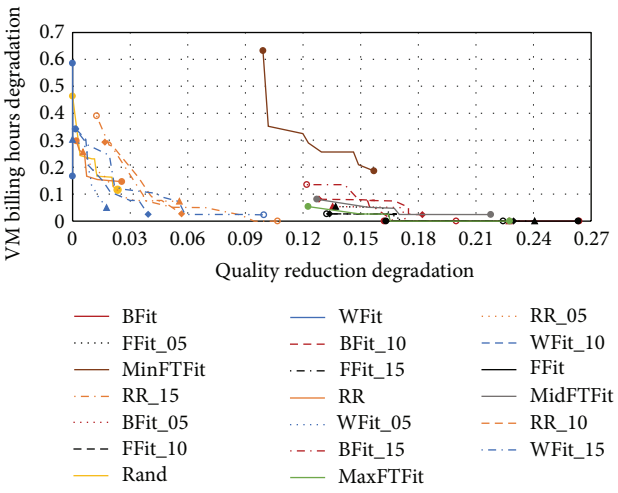


Figure 9: Pareto fronts.

Finally, WFit_15 has a wide range of solutions for QRD (from 0.019 to 0.099) but only 20% of its solutions are over the 20% of cost degradation.

WFit versions cover different sectors in the Pareto front, and they show the best compromise between both objectives for the twenty strategies.

The MaxFTFit solution space is in the same range for cost as Bfit and FFit. It overcomes the quality reduction of both strategies.

WFit_05, WFit_10, WFit_15, and MaxFTFit strategies cover better the solution space and Pareto front. They are good options for the VoIP providers. Figure 9 shows the twenty approximations of Pareto fronts generated by the studied strategies.

Using the set coverage metric, described in Section 6.2, two sets of nondominated solutions can be compared. The rows of Table 7 show the values SC(A, B) for the dominance of strategy A over strategy B. The columns indicate SC(B, A), that is, dominance of B over A. The last two columns show the average of SC(A, B) for row A over column B and ranking based on the average dominance. Similarly, the last two rows show average dominance of B over A and rank of the strategy in each column.

Table 7: Set coverage and ranking (days).

A	*B*																				
	BFit	BFit_05	BFit_10	BFit_15	FFit	FFit_05	FFit_10	FFit_15	MaxFTFit	MidFTFit	MinFTFit	Rand	RR	RR_05	RR_10	RR_15	WFit	WFit_05	WFit_10	WFit_15	Mean	Rank
BFit	1	0	0	0	0.30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.065	18
BFit_05	0.07	1	0.06	0.02	0.08	0.17	0.05	0.01	0.01	0.01	0	0	0	0	0	0	0	0	0	0	0.074	17
BFit_10	0.03	0.17	1	0.07	0.02	0.11	0.10	0.05	0.02	0.02	0	0	0	0	0	0	0	0	0	0	0.079	16
BFit_15	0.03	0.23	0.28	1	0.05	0.10	0.11	0.10	0.03	0.01	0	0	0	0	0	0	0	0	0	0	0.096	14
FFit	0.21	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.061	19
FFit_05	0.33	0.06	0.04	0.02	0.30	1	0	0	0.01	0	0	0	0	0	0	0	0	0	0	0	0.088	15
FFit_10	0.18	0.17	0.09	0.06	0.18	0.38	1	0.06	0.04	0.03	0	0	0	0	0	0	0	0	0	0	0.109	11
FFit_15	0.21	0.29	0.19	0.12	0.21	0.33	0.34	1	0.03	0.06	0	0	0	0	0	0	0	0	0	0	0.139	8
MaxFTFit	0.43	0.42	0.28	0.24	0.46	0.60	0.47	0.38	1	0.09	0	0	0	0	0	0	0	0	0	0	0.218	2
MidFTFit	0.02	0.20	0.31	0.18	0	0.14	0.13	0.08	0.03	1	0	0	0	0	0	0	0	0	0	0	0.105	13
MinFTFit	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.050	20
Rand	0	0	0	0	0	0	0	0	0	0.01	0.94	1	0.01	0.09	0.03	0	0	0.05	0.01	0.02	0.107	12
RR	0	0	0	0	0	0	0	0.01	0	0	0.90	0.52	1	0.10	0.02	0.03	0	0.07	0.00	0.02	0.133	10
RR_05	0	0.04	0.04	0.04	0	0	0	0.01	0.01	0.08	0.97	0.03	0.02	1	0.21	0.11	0.01	0.01	0.19	0.14	0.145	7
RR_10	0.01	0.13	0.20	0.17	0	0.06	0.06	0.07	0.04	0.28	0.99	0.02	0	0.02	1	0.31	0	0	0.01	0.21	0.178	6
RR_15	0.02	0.32	0.39	0.39	0.02	0.11	0.17	0.16	0.06	0.52	0.98	0	0	0.02	0.01	1	0	0	0	0.02	0.210	3
WFit	0	0	0	0	0	0	0	0	0	0	0.84	0.27	0.41	0.08	0.01	0.02	1	0.09	0.03	0.01	0.138	9
WFit_05	0	0.04	0.02	0.02	0	0.02	0	0.01	0	0.03	0.96	0.42	0.15	0.51	0.20	0.15	0.01	1	0.22	0.15	0.195	4
WFit_10	0	0.09	0.15	0.17	0.01	0.05	0.06	0.06	0.03	0.25	0.97	0.01	0.04	0.06	0.38	0.28	0	0.02	1	0.28	0.194	5
WFit_15	0.02	0.26	0.32	0.32	0.01	0.14	0.16	0.14	0.08	0.45	0.98	0	0.01	0.02	0.06	0.39	0	0	0.02	1	0.219	**1**
Mean	0.128	0.171	0.169	0.140	0.132	0.160	0.132	0.106	0.069	0.142	0.477	0.113	0.082	0.095	0.096	0.114	0.051	0.062	0.074	0.092		
Rank	12	19	18	15	14	17	13	9	3	16	20	10	5	7	8	11	**1**	2	4	6		

The ranking of the strategies is based on the coverage percentage. The higher ranking implies that the front is better.

Table 7 reports the SC results for each of the twenty Pareto fronts. According to the set coverage metric, the strategy that has the best compromise between the number of billing hours and quality reduction is Wfit_15, followed by MaxFTFit, RR_15, and WFit_05.

We see that MaxFTFit dominates the fronts of other strategies in the range of 0–60%, with 21.8% in average occupying the second rank. SC(A, MaxFTFit) shows that MaxFTFit is dominated by other fronts on 6.9% in average. Meanwhile, WFit_05 and MaxFTF with the second and third ranks are dominated by other strategies on 19.5% and 21.9% on average, respectively. They are dominated for other strategies on 6.2% and 6.9%.

However, we should not consider only Pareto fronts, when many solutions are outside the Pareto optimal solutions. This is the case of BFit_xx, FFit_xx, and RR_xx: although the Pareto fronts are of good quality, many of the generated solutions are quite far from them, and, hence, a single run of the algorithm may produce significantly worse results.

11. Conclusions and Future Work

In this paper, we formulate and study scheduling problems addressing VoIP service in cloud computing. We define models of the provider cost and quality of service and propose new on-line nonclairvoyant bin-packing algorithms for call allocation. Unlike the standard formulation of the problem, our bins are always open, even if they are completely packed. Items in bins can disappear after call termination, and utilization can be changed at any moment. The problem is dynamic, when no knowledge about call duration or its estimation is used.

Due to the fact that VM parameters are changing over time, traditional scheduling techniques based on number of calls do not adapt well to this dynamism. VoIP solutions do not take into account uncertainty in dynamic and unpredictable cloud environments.

Our approach is suitable for environment with presence of uncertainty. It takes allocation decisions depending on the actual cloud and VM characteristics at the moment of allocation such as number of available virtual machines and their utilization. It can cope with different workloads, type of calls (voice, video, conference, etc.), cloud properties, and cloud uncertainties, such as elasticity, performance changing, virtualization, loosely coupling application to the infrastructure, and parameters such as an effective processor speed and actual number of available virtual machines. We propose twenty VoIP scheduling strategies and evaluate their performance by comprehensive simulation analysis on real data considering six months of the MIXvoip company service.

We show that the proposed algorithms can be efficiently used in a VoIP cloud environment. The monoobjective and biobjective analyses provide a good compromise between saving money and voice quality.

However, further study is required to assess their actual performance and effectiveness in a real domain. This will be the subject of future work. Moreover, quality in communication systems, hypervisor-level scheduling, dynamic consolidation of calls and VMs, and distributed load balancing are other important issues to be addressed.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is partially supported by CONACYT (Consejo Nacional de Ciencia y Tecnología, México) (Grant no. 178415). The work of Pascal Bouvry is partly funded by INTER/CNRS/11/03 Green@Cloud. The work of Alexander Yu. Drozdov is partially supported by the Ministry of Education and Science of Russian Federation under Contracts RFMEFI58214X0003 and 02.G25.31.0061/12/02/2013 (Government Regulation no. 218 from 09/04/2010).

References

[1] A. M. Alakeel, "A guide to dynamic load balancing in distributed computer systems," *International Journal of Computer Science and Network Security*, vol. 10, no. 6, pp. 153–160, 2012.

[2] A. M. Simionovici, A. A. Tantar, P. Bouvry, A. Tchernykh, J. M. Cortés-Mendoza, and L. Didelot, "VoIP traffic modelling using gaussian mixture models, Gaussian processes and interactive particle algorithms," in *Proceedings of the 4th IEEE International Workshop on Cloud Computing Systems, Networks, and Applications (CCSNA '15), in Conjunction with IEEE Global Communications Conference (GLOBECOM '15)*, San Diego, Calif, USA, December 2015.

[3] J. M. Cortés-Mendoza, A. Tchernykh, A.-M. Simionovici et al., "VoIP service model for multi-objective scheduling in cloud infrastructure," *International Journal of Metaheuristics*, vol. 4, no. 2, pp. 185–203, 2015.

[4] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[5] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, and E. Talbi, "Towards understanding uncertainty in cloud computing resource provisioning," *Procedia Computer Science*, vol. 51, pp. 1772–1781, 2015.

[6] MIXvoip S.a. Company, https://www.mixvoip.com/.

[7] P. Montoro and E. Casilari, "A comparative study of VoIP standards with asterisk," in *Proceedings of the 4th International Conference on Digital Telecommunications (ICDT '09)*, pp. 1–6, Colmar, France, July 2009.

[8] "3CX Phone System and ATOM N270 Processor Benchmarking," http://www.3cx.com/blog/voip-howto/atom-processor-n270-benchmarking/.

[9] Cisco Systems, Understanding Codecs: Complexity, Hardware Support, MOS, and Negotiation, http://www.cisco.com/c/en/us/support/docs/voice/h323/14069-codec-complexity.html.

[10] L. Madsen, R. Bryant, and J. V. Meggelen, *Asterisk: The Definitive Guide*, O'Reilly Media, 3rd edition, 2011.

[11] J. So, "Scheduling and capacity of VoIP services in wireless OFDMA systems," in *VoIP Technologies*, S. Kashihara, Ed., chapter 11, pp. 237–252, InTech, Rijeka, Croatia, 2011.

[12] H. Lee, T. Kwon, D.-H. Cho, G. Lim, and Y. Chang, "Performance analysis of scheduling algorithms for VoIP services in IEEE 802.16e systems," in *Proceedings of the IEEE 63rd Vehicular Technology Conference (VTC '06)*, pp. 1231–1235, Melbourne, Australia, July 2006.

[13] M. Folke, S. Landström, U. Bodin, and S. Wänstedt, "Scheduling support for mixed VoIP and web traffic over HSDPA," in *Proceedings of the IEEE 65th Vehicular Technology Conference (VTC-Spring '07)*, pp. 814–818, IEEE, Dublin, Ireland, April 2007.

[14] N. Bayer, B. Xu, V. Rakocevic, and J. Habermann, "Application-aware scheduling for VoIP in wireless mesh networks," *Computer Networks*, vol. 54, no. 2, pp. 257–277, 2010.

[15] S. Wu, L. Zhou, D. Fu, H. Jin, and X. Shi, "A real-time scheduling framework based on multi-core dynamic partitioning in virtualized environment," in *Network and Parallel Computing: 11th IFIP WG 10.3 International Conference, NPC 2014, Ilan, Taiwan, September 18–20, 2014. Proceedings*, vol. 8707 of *Lecture Notes in Computer Science*, pp. 195–207, Springer, Berlin, Germany, 2014.

[16] A. Mazalek, Z. Vranova, and E. Stankova, "Analysis of the impact of IPSec on performance characteristics of VoIP networks and voice quality," in *Proceedings of the 5th International Conference on Military Technologies (ICMT '15)*, pp. 1–5, IEEE, Brno, Czech Republic, May 2015.

[17] L. R. Costa, L. S. Nunes, J. L. Bordim, and K. Nakano, "Asterisk PBX capacity evaluation," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW '15)*, pp. 519–524, IEEE, Hyderabad, India, May 2015.

[18] K. Cheng, Y. Bai, R. Wang, and Y. Ma, "Optimizing soft real-time scheduling performance for virtual machines with SRT-xen," in *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid '15)*, pp. 169–178, Shenzhen, China, May 2015.

[19] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.

[20] A. Tchernykh, L. Lozano, U. Schwiegelshohn et al., "Online bi-objective scheduling for IaaS clouds ensuring quality of service," *Journal of Grid Computing*, vol. 14, no. 1, pp. 5–22, 2016.

[21] A. Lezama Barquet, A. Tchernykh, and R. Yahyapour, "Performance evaluation of infrastructure as a service clouds with SLA constraints," *Iberoamerican Journal of Research Computing and Systems*, vol. 17, no. 3, pp. 401–411, 2013.

[22] "CloudSim: a framework for modeling and simulation of Cloud Computing infrastructures and services," http://www.cloudbus.org/cloudsim/.

[23] T. D. Dang, B. Sonkoly, and S. Molnár, "Fractal analysis and modeling of VoIP traffic," in *Proceedings of the 11th International Telecommunications Network Startegy and Planning Symposium*, pp. 123–130, Vienna, Austria, June 2004.

[24] J. M. Cortés-Mendoza, A. Tchernykh, A. Drozdov, P. Bouvry, A. Simionovici, and A. Avetisyan, "Distributed adaptive VoIP load balancing in hybrid clouds," in *Proceedings of the 1st Russian Conference on Supercomputing (RuSCDays '15)*, pp. 676–686, Moscow, Russia, September 2015.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 6208358, 13 pages
http://dx.doi.org/10.1155/2016/6208358

Research Article

Task Classification Based Energy-Aware Consolidation in Clouds

HeeSeok Choi,[1] JongBeom Lim,[2] Heonchang Yu,[1] and EunYoung Lee[3]

[1]*Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea*
[2]*IT Convergence Education Center, Dongguk University, Seoul, Republic of Korea*
[3]*Department of Computer Science, Dongduk Women's University, Seoul, Republic of Korea*

Correspondence should be addressed to EunYoung Lee; elee@dongduk.ac.kr

Received 22 January 2016; Accepted 3 August 2016

Academic Editor: Zhihui Du

We consider a cloud data center, in which the service provider supplies virtual machines (VMs) on hosts or physical machines (PMs) to its subscribers for computation in an on-demand fashion. For the cloud data center, we propose a task consolidation algorithm based on task classification (i.e., computation-intensive and data-intensive) and resource utilization (e.g., CPU and RAM). Furthermore, we design a VM consolidation algorithm to balance task execution time and energy consumption without violating a predefined service level agreement (SLA). Unlike the existing research on VM consolidation or scheduling that applies none or single threshold schemes, we focus on a double threshold (upper and lower) scheme, which is used for VM consolidation. More specifically, when a host operates with resource utilization below the lower threshold, all the VMs on the host will be scheduled to be migrated to other hosts and then the host will be powered down, while when a host operates with resource utilization above the upper threshold, a VM will be migrated to avoid using 100% of resource utilization. Based on experimental performance evaluations with real-world traces, we prove that our task classification based energy-aware consolidation algorithm (TCEA) achieves a significant energy reduction without incurring predefined SLA violations.

1. Introduction

Nowadays, cloud computing has become an efficient paradigm of offering computational capabilities as a service based on a pay-as-you-go model [1] and many studies have been conducted in diverse cloud computing research areas, such as fault tolerance and quality of service (QoS) [2, 3]. Meanwhile, virtualization has been touted as a revolutionary technology to transform cloud data centers (e.g., Amazon's elastic compute cloud and Google's compute engine) [4]. By taking advantage of the virtualization technology, running cloud applications on virtual machines (VMs) has become an efficient solution of consolidating data centers because the utilization rate of data centers has been found to be low, typically ranging from 10 to 20 percent [5]. In other words, a single host (physical machine) can run multiple VMs simultaneously and VMs can be relocated dynamically by live migration operations, leading to high resource utilization. Another issue of data centers is high energy consumption, which results in substantial carbon dioxide emissions (about 2 percent of the global emissions). A typical data center consumes as much energy as 25,000 households do [6]. In this regard, an efficient energy consumption strategy in nonvirtualization environments (smart grids) has been carried out [7].

As the virtualization technology [8, 9] has become popular widely, organizations or companies began to build their own private cloud data centers using commodity hardware. In this regard, there exists a need for designing more efficient and effective VM consolidation techniques to reduce energy consumption in cloud data centers. The simplest way to achieve energy reduction in cloud computing environments is to minimize the number of physical machines (PMs) by allocating more VMs in a PM. However, this solution may lead to a high degree of service level agreement (SLA) violations when each VM requires the host's limited resources. Moreover, the relationship between CPU utilization and power consumption is not linear as shown in Figure 1. The power consumption of CPU increases more than linearly as utilization increases. More importantly, when the CPU utilization is above 90%, the power consumption jumps up quickly due to the architectural design and turbo boost

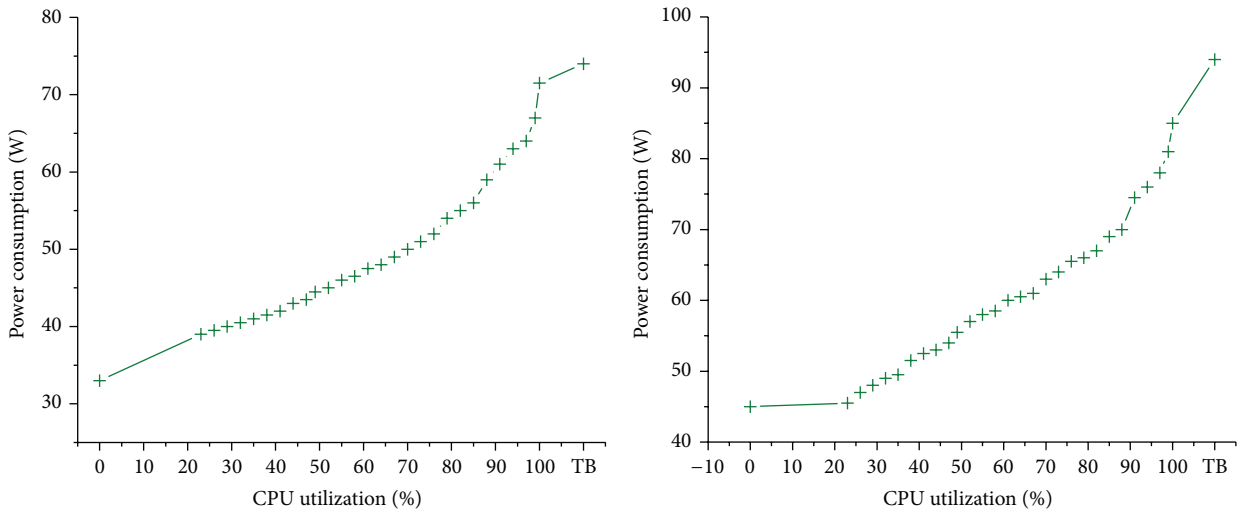


Figure 1: Energy consumption of i5 and i7 CPUs (TB indicates turbo boost).

feature. In other words, the performance to power ratio [10] exhibits sublinear growth, and therefore, just putting many VMs to a PM utilizing 100% of CPU is not always the best solution in terms of performance, energy consumption, and SLA violations. We take Intel i5 and i7 CPUs in our experiments, rather than server class CPUs in Figure 1, because, for small and medium sized companies, using commodity hardware like Intel i5 or i7 to build a private cloud is more affordable and accessible [11].

In this paper, we present a new VM and task consolidation mechanism in cloud computing environments. The proposed method is based on task classification, in which we divide cloud tasks into two categories: computation-intensive and data-intensive tasks. A computation-intensive task refers to a computation-bounded application program. Such applications devote most of their execution time to fulfill computational requirements as opposed to I/O and typically require small volumes of data, while a data-intensive task refers an I/O-bounded application with a need to process large volumes of data. Such applications devote most of their processing time to I/O, movement, and manipulation of data. The basic idea of our approach is twofold. One is that when we need to migrate cloud tasks due to a migration policy, we favor a computation-intensive task for migration rather than a data-intensive task since the migration time for computation-intensive tasks is shorter than that of data-intensive tasks. In order to migrate data-intensive tasks, it is necessary to move data for processing as well, and this transferring of data generates communication overheads. Then, we prefer the target VM with no computation-intensive tasks because data-intensive tasks consume less CPU resources, thereby providing a comfortable executing environment for the computation-intensive task. The other is to use a double threshold approach (i.e., upper threshold and lower threshold) for VM migrations and optimization. When a VM's utilization is either above the upper threshold or below the lower threshold, the VM is scheduled for migration. Our double threshold approach is different from previous work in that no algorithm is proposed to use the upper and lower thresholds simultaneously in an effective way to the best of our knowledge. With an extensive measurement observation, we identified that there is much room for optimization by balancing performance and energy consumption.

Our work differs from traditional scheduling algorithms in the literature by designing and implementing a novel consolidation mechanism based on a task classification approach. We develop corresponding task scheduling and VM allocation algorithms for cloud tasks executed in virtualized data centers.

The major contributions of this paper are summarized as follows:

(i) We designed an energy-aware cloud data center consolidation mechanism based on task classification, while preserving performance and SLA guarantee.

(ii) We developed a cloud task scheduling and VM allocation algorithms that solve problems about when and how to migrate tasks and VMs in an energy efficient way.

(iii) We formulated a double threshold algorithm for further optimization to improve the performance to power ratio.

(iv) We undertook a comprehensive analysis and performance evaluation based on real-world workload traces.

The rest of this paper is organized as follows. Section 2 describes our research motivation and our intuition for consolidation in virtualized clouds. In Section 3, the task classification based energy-aware consolidation scheduling mechanism and the main principles behind it are presented. The experiments and performance analysis are given in Section 4. The related work in the literature is summarized in Section 5. Finally, Section 6 concludes the paper.

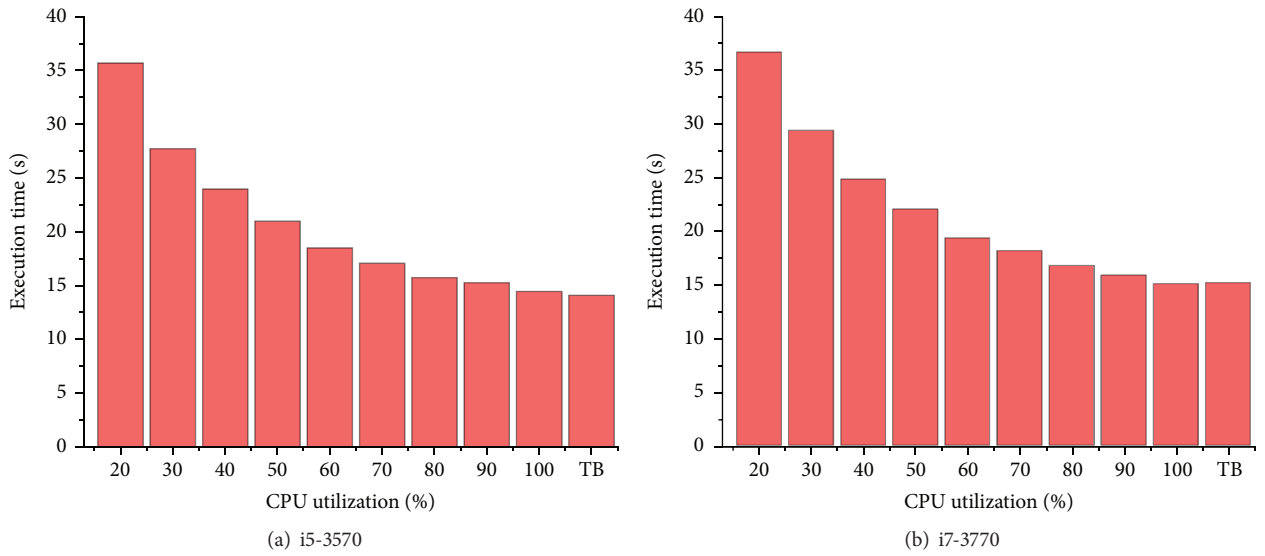


Figure 2: Energy consumption and execution time of matrix multiplication of i5 and i7 CPUs.

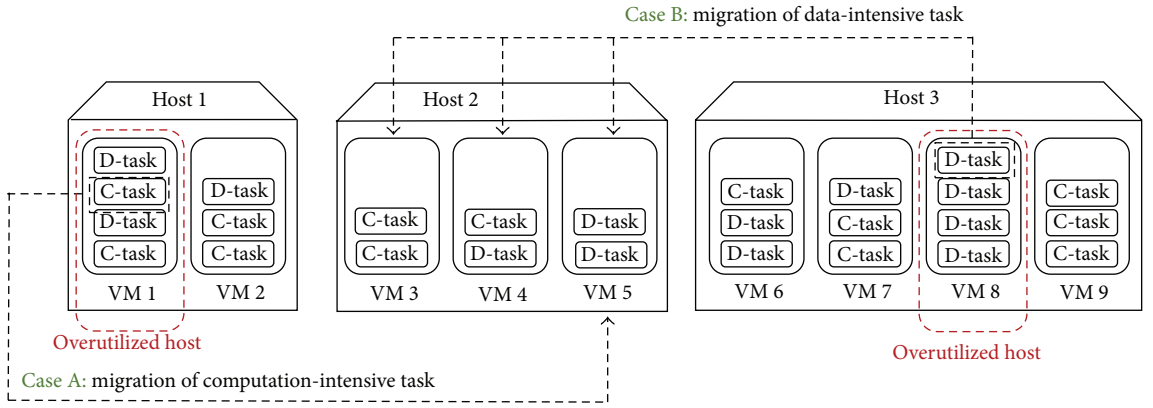


Figure 3: An illustrative example of TCEA.

2. Motivation and the Basic Idea

As the virtualization technology has been widely used, it is easily possible to construct a private cloud computing environment with open-source infrastructure as a service (IaaS) solutions and commodity hardware (e.g., desktop-level CPUs and peripherals). Figure 2 shows execution time of a matrix multiplication benchmark program and its performance to power ratio with CPU utilization for Intel i5-3570 and i7-3770 CPUs. With CPU utilization below 50%, the performance gain from the CPUs is noticeable as CPU utilization increases as the performance to power ratio indicates. However, when CPU utilization is above 50% the performance to power ratio grows sublinearly. This means that using high CPU utilization is not always an energy-efficient way to perform tasks. Even when we use a turbo boost feature, one of dynamic voltage and frequency scaling (DVFS) techniques, the performance gain of high frequency of CPU operations is not big considering the performance to power ratio.

Hence, we devise another approach using a threshold of CPU utilization so that a host that manages a couple of VMs does not exceed a predefined CPU utilization threshold. When a host exceeds the threshold, our consolidation algorithm determines to migrate one of the tasks or VMs on the host to another as depicted in Figure 3. Each task is categorized as C-task (computation-intensive task) or D-task (data-intensive task) and is assumed to use 25% of resources or utilization for a VM for simplicity in this example. Note that the task categorization mechanism of C-task and D-task is explained in the next section. Assuming that the threshold is 75% for a VM, tasks in VM 1 and VM 8 should be migrated to underutilized VMs. For Case A, in which there are C-tasks and D-tasks in a VM, our consolidation algorithm chooses a C-task to be migrated and preferentially selects a target VM with no C-tasks since migrating a C-task takes much shorter time compared to a D-task and migrating a D-task introduces a major I/O bottleneck in the host. For Case B, in which there are only D-tasks but C-tasks, we only consider underutilized

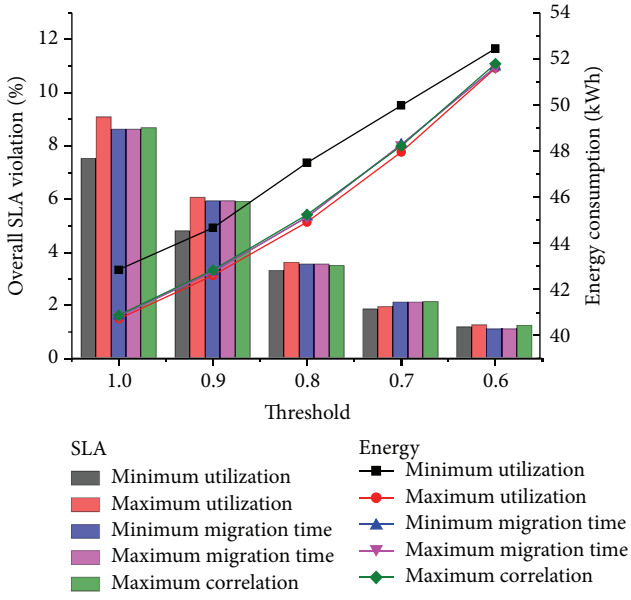


Figure 4: Energy consumption and SLA violations with threshold and migration policies.

VMs for target, disregarding the category of tasks running on the target VM. For task migration, there are many prevalent software and management technologies, such as openMosix, which is a Linux kernel extension that allows processes to migrate to other nodes seamlessly.

On the other hand, choosing a proper threshold value is an important factor that influences the overall performance and there is a tradeoff between the threshold value and SLA violation. Figure 4 shows the tradeoff with various migration policies. Obviously, lowering the threshold value leads to lower energy consumption, but it causes SLA violations, meaning a user's request for tasks cannot guarantee to be succeeded in preagreed metrics. In a condensed situation, where there is no host that can afford additional VMs and the ratio of PM to VM is low, it is more desirable to use a higher threshold value, whereas, in a sparse situation, where there are many free hosts available for additional VMs and the ratio of PM to VM is high, it allows having a lower threshold value but it is energy consuming and wastes resources. As far as the latter case is concerned, we use a double threshold approach to reduce energy consumption more, while incurring the overall SLA violation as little as possible. The resource types for a system are CPU, memory, storage, network, and so forth. Among them, CPU is the most dominant factor that influences energy consumption [12]. In this paper, we focus on CPU utilization for migration policies and leave integrating other types of resource into the migration policies as future work.

3. Task Classification Based Energy-Aware Consolidation Algorithm (TCEA)

As shown in Figure 5, we consider a typical cloud data center with a cloud portal. When a user submits a task to the cloud portal, TCEA first performs a task classification process based on configurations of the task and historical logs. The task is categorized as either computation-intensive or data-intensive. Then, with this task classification information, we assign the task to an appropriate VM and consolidate VMs in the data center in an energy-aware way. After that, TCEA periodically checks hosts with a predefined threshold value so that unnecessary hosts are powered down after migrating their VM to others, while maintaining SLA. The detailed description of our proposed algorithms is given below.

(A) Double Threshold Scheme. Our consolidation algorithms are based on the double threshold scheme. In order to save energy consumption of a cloud data center, one may consider using the minimum number of hosts by utilizing CPU as much as possible for VMs. However, this approach is not an energy efficient solution because it disregards the performance to power ratio. Thus, TCEA uses the upper threshold to prevent heating CPUs up. On the other hand, when many of the hosts are easygoing as a whole, it is necessary to minimize active hosts to save superfluous energy consumption by consolidating VMs. For that purpose, we employ the lower threshold. With the lower threshold, TCEA periodically checks hosts and VMs whether it requires VM or task consolidation. For example, if a host operates with CPU utilization below the lower threshold, we migrate VMs on the host to other hosts as long as there are available hosts to accommodate the VMs without restricting VMs' liberty. With these in mind, it is important to choose proper values for the double threshold scheme, that is, the upper threshold and lower threshold, considering the tradeoff between performance and energy consumption. To determine the conditions of suitable threshold values, we conduct several experiments in Section 4.

(B) Task Classifier. Unlike previous work, we consider a task's characteristics in consolidating a cloud data center. Towards this end, we place a task classifier module to categorize tasks into computational-intensive or data-intensive tasks. When a user submits a task, it examines history log files to check whether it has been performed before. If so, TCEA uses the previous classification information without performing the task classification process. If not, it performs the task classification process as shown in Algorithm 1.

The criteria of classifying tasks in the task classifier function are based on the communication to computation ratio [13]. By examining the execution time and task transfer time of a task, it puts the task to the corresponding queue. In other words, when computation time is greater than task transfer time of a task, the task classifier makes the task resident in $taskQueue_{computation}$. Otherwise, the task is considered as data-intensive. The classification information of the task is also stored in the storage for future use.

(C) Task Assignment. The next step after performing the task classification process is to assign tasks to appropriate VMs. When assigning a task, TCEA first tries to find a host whose utilization is relatively low as shown in Algorithm 2. Then, it checks all the VMs in the host by counting the number

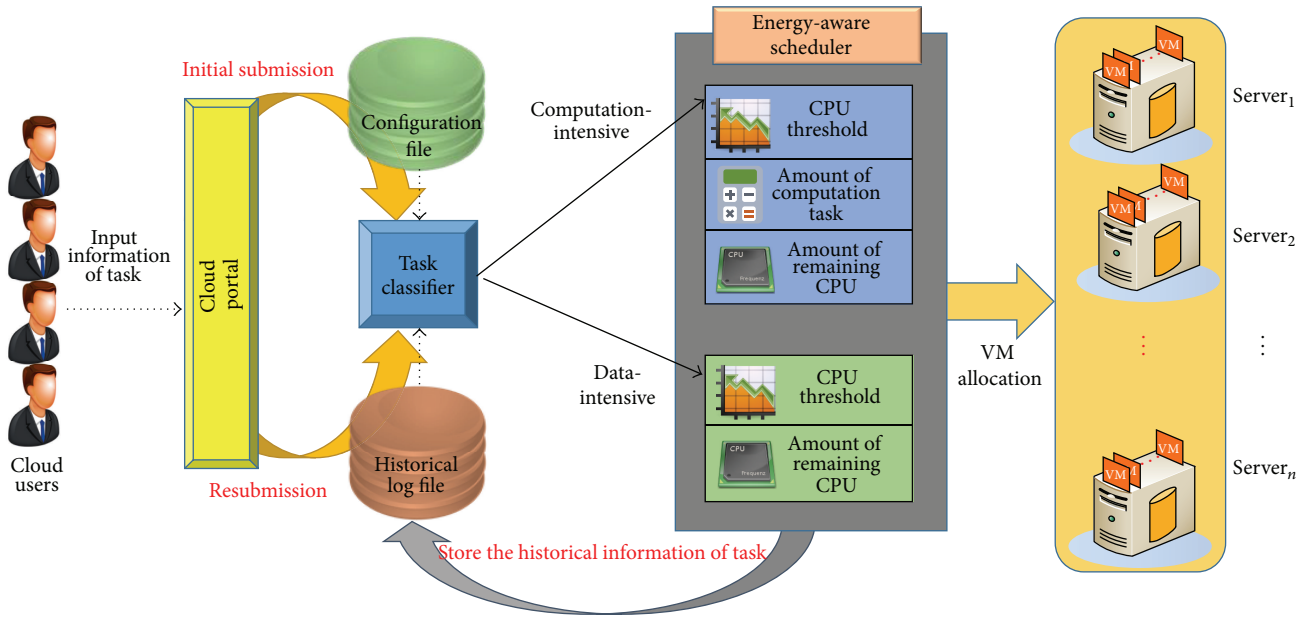


Figure 5: System architecture of TCEA.

of computation-intensive tasks. Out of the VMs, a VM that has the least number of computation-intensive tasks can be a candidate when the task is computation-intensive. After iterating this phase, the task assignment function selects a VM for the task.

When the type of a task is data-intensive, TCEA does not care about the types of tasks for finding target VMs. The only consideration is the number of tasks running in VMs. Thus, it finds a VM that runs the minimum number of tasks in order to balance the load. For optimization, the task assignment function migrates a task to another VM. At this stage, we favor computation-intensive tasks for migration because migrating data-intensive tasks is inefficient. In other words, migrating data-intensive tasks takes more time than migrating computation-intensive tasks since it is necessary to move the data of data-intensive tasks as well. When finding an overutilized host, TCEA prefers a VM that runs the largest number of computation-intensive tasks for migration. This is based on the fact that migrating a computation-intensive task is more efficient than migrating a data-intensive task with regard to the number of migrations and utilization shifting. Once a task is chosen for migration, the next step is to choose a target VM. There are two conditions for choosing a target VM. One is CPU utilization and the other is the number of computation-intensive tasks. Among VMs whose host's CPU utilization is low, a VM that runs the least number of computation-intensive tasks will be chosen for the target VM. Then, the task is scheduled to be migrated accordingly.

(D) Consolidation of VMs. For VM consolidation, it is essential to handle and manage VMs and hosts chosen by the double threshold scheme. Algorithm 3 shows the VM consolidation in TCEA in detail. When a host's utilization is above the upper threshold (i.e., overutilized hosts), TCEA chooses a VM to be migrated considering the number of computation-intensive tasks. The more computation tasks a VM has, the more likely the VM is to be a source for migration. Once a source VM is selected, a target host selection phase is performed. Since a source VM will occupy a large portion of utilization, it is preferable to choose a target host whose utilization is relatively low. Therefore, the chosen target host may have fewer numbers of computation-intensive tasks than others. On the other hand, when managing underutilized hosts chosen by the lower threshold, all the VMs in the host will be migrated to hosts whose utilization is normal across the data center. The reason why TCEA chooses normally utilized hosts as migration targets is to exploit the performance to power ratio. Choosing a host of full utilization as a target will result in more energy consumption and consolidation management overheads. For example, when a host becomes overutilized and is chosen as a target host, TCEA will perform redundant load balancing operations.

(E) Task Classification Based Energy-Aware Consolidation Algorithm (TCEA). Algorithm 4 covers our overall consolidation and scheduling scheme. Note that the procedure of lines (1)–(6) is triggered upon receipt of a set of tasks and that of lines (7)–(18) is performed periodically. The task classifier function and the task assignment function are responsible for consolidation and management of tasks in TCEA. TCEA monitors VMs and hosts in the cloud data center for status updates. With the predefined values including the upper and lower thresholds, TCEA maintains $list_{upper}$, $list_{normal}$, and $list_{lower}$ of hosts. To balance performance and energy consumption, VMs in $list_{upper}$ and $list_{lower}$ will be migrated to $list_{normal}$. It is worth noting that choosing the proper values of the upper threshold, lower threshold, and the number of

```
(1)  if task_i has no historical log file
(2)     if VM execution time is greater than data movement time
(3)        taskQueue_Computation ← taskQueue_Computation ∪ task_i;
(4)     else
(5)        taskQueue_Data ← taskQueue_Data ∪ task_i;
(6)     end if
(7)  else // The task_i has historical log file
(8)     Retrieve information from the configuration file;
(9)     Classify data type using obtained information;
(10) end if
```

Algorithm 1: *Task_Classifier ( )*.

```
(1)  if task_i ∈ taskQueue_Computation
(2)     for all host_i ∈ list_normal, ∀i ∈ {1, 2, ..., n};
(3)        Find a host_i with the lowest CPU utilization;
(4)           for all vm_i ∈ host_i, ∀i ∈ {1, 2, ..., n};
(5)              Check the number of computation-intensive tasks;
(6)              Find a vm_i having the least number of computation-intensive tasks;
(7)           end for
(8)     end for
(9)     Assign task_i to vm_i
(10) else if task_i ∈ taskQueue_Data
(11)    for all host_i ∈ list_normal, ∀i ∈ {1, 2, ..., n};
(12)       Find a host_i with the lowest CPU utilization;
(13)          for all vm_i host_i, ∀i ∈ {1, 2, ..., n};
(14)             Check the number of tasks;
(15)             Find a vm_i having the least number of tasks;
(16)          end for
(17)    end for
(18)    Assign task_i to vm_i;
(19) end if
```

Algorithm 2: *Assign_Task ( )*.

```
(1)  // for over-utilized hosts ∈ list_upper
(2)     Find a host_i with the highest CPU utilization ∈ list_upper;
(3)     for all vm_i ∈ host_i, ∀i ∈ {1, 2, ..., n};
(4)        Check the number of computation-intensive tasks;
(5)        Find a vm_i having the largest number of computation-intensive tasks;
(6)     end for
(7)     for all host_j ∈ list_normal;
(8)        Check the number of computation-intensive tasks;
(9)        Find a host_j having the least number of computation-intensive tasks;
(10)    end for
(11)    Migrate vm_i to host_j;
(12) // for under-utilized hosts ∈ list_lower
(13)    for all host_j ∈ list_lower, ∀j ∈ {1, 2, ..., n};
(14)       Find a host_j with the lowest CPU utilization;
(15)    end for
(16)    Migrate all VMs ∈ host_j to host_k ∈ list_normal;
(17)    Switch off host_j;
```

Algorithm 3: *Consolidate_VM ( )*.

(1) for all $task_i$, where $task_i \in$ Task, $\forall i \in \{1, 2, \ldots, n\}$;
(2) *Task_Classifier ( )*
(3) *Assign_Task ( )*
(4) end for
(5) Update the status of each task;
(6) Store monitored status information;
(7) for all $host_i$, where $host_i \in$ Host, $\forall i \in \{1, 2, \ldots, n\}$;
(8) Monitor the status of host;
(9) if CPU utilization is higher than $threshold_{upper}$
(10) $list_{Upper} \leftarrow list_{Upper} \cup host_i$;
(11) else if CPU utilization is lower than $threshold_{lower}$
(12) $list_{lower} \leftarrow list_{lower} \cup host_i$;
(13) else
(14) $list_{normal} \leftarrow list_{normal} \cup host_i$;
(15) end if
(16) Store monitored status information;
(17) end for
(18) *Consolidate_VM ( )*

Algorithm 4: Task classification based energy-aware consolidation algorithm.

VMs to be migrated influences the performance of TCEA. In the next section, we validate TCEA for energy efficiency and performance with these parameters.

4. Performance Evaluation

In this section, we present experimental results that demonstrate the performance of TCEA for reducing energy consumption by managing VM consolidation while achieving SLA satisfaction. As input, we use real task traces (Intel Netbatch logs [14]) and artifact task logs for a fixed combination of computation-intensive tasks and data-intensive tasks. For experiments, we assume that there are 50 hosts and 100 VMs running in the cloud data center unless specified otherwise. A host is equipped with a quad-core CPU (i7-3770) with 4 GB of RAM and gigabit Ethernet. A user can specify the type of a VM such as the number of vCPU, RAM, and storage capacity. Otherwise, a default VM setting with 1 GB of RAM and 1 vCPU is used.

In this experiment, we analyze the runtime of TCEA with varying upper thresholds from 100% to 60%. We conduct this experiment for the real world datasets mentioned above. In Figure 6, x-axis denotes the upper threshold and y-axis represents the energy consumption, the number of VM migrations, and the number of host shutdowns. The number of VM migrations and the number of host shutdowns are constantly going down as the upper threshold decreases. With decreased upper threshold, the available hosts tend to remain alive because VMs should reside in hosts whose utilization is below the upper threshold, and therefore, the number of VM migrations is reduced as well. For energy consumption, 90% is optimal. This indicates that (1) although hosts with 100% of upper threshold maintain more VMs, 100% is not the best threshold due to the performance to power ratio, (2) even though the number of host shutdowns peaks with

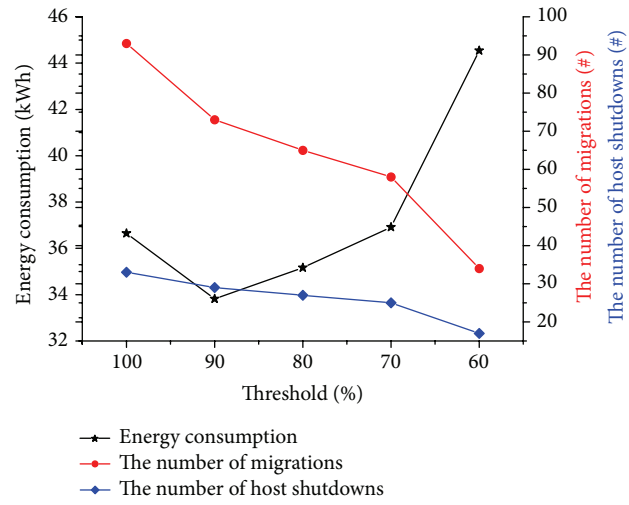


Figure 6: Performance results for upper threshold.

100% of upper threshold, the energy reduction of using the lower threshold (90%) dominates that of the number of host shutdowns, and (3) the number of VM migrations decreases with lower upper threshold because the probability of finding satisfactory target VMs gets lower too. For the rest of experiments, we use 90% of upper threshold unless specified otherwise.

For a sparse situation, where there are many free hosts available for additional VMs and the ratio of PM to VM is high, we devise an optimization algorithm to migrate VMs from underutilized hosts to others and shutdown the hosts, thereby reducing energy consumption. To this end, we use a lower threshold such that VMs in a host below the lower threshold are scheduled to be migrated to other hosts, and then the host gets shutdown. Figure 7 shows energy consumption, the number of VM migrations, and the number of host shutdowns with varying lower thresholds (e.g., 0.8 of x-axis means that 20% of hosts are chosen by the lower threshold). Comparing with default (no task classification is performed), TCEA consumes 14.05% less energy on average. When the lower threshold is 50%, the difference between default and TCEA reaches a peak. With respect to energy consumption, the number of VM migrations, and the number of host shutdowns, we use 50% of lower threshold for the rest of experiments unless specified otherwise.

To verify the effectiveness of lower thresholds, we conduct another experiment showing energy consumption, the number of VM migrations, and the number of host shutdowns with VM ratios by increasing the number of VMs and hosts (1x means a default setting of 100 VMs and 50 hosts). Note that, in this experiment, 0.9 of VM ratio means that 10% of hosts whose utilization is below the lower threshold are scheduled to be powered down by migrating their VMs. As shown in Figure 8, around 50% of the VM ratio suits our purpose in terms of energy consumption, the number of VM migrations, the number of host shutdowns, and SLA violations. The ratio below 0.5 leads to SLA violations; therefore we do not use ratio lower than 0.5.

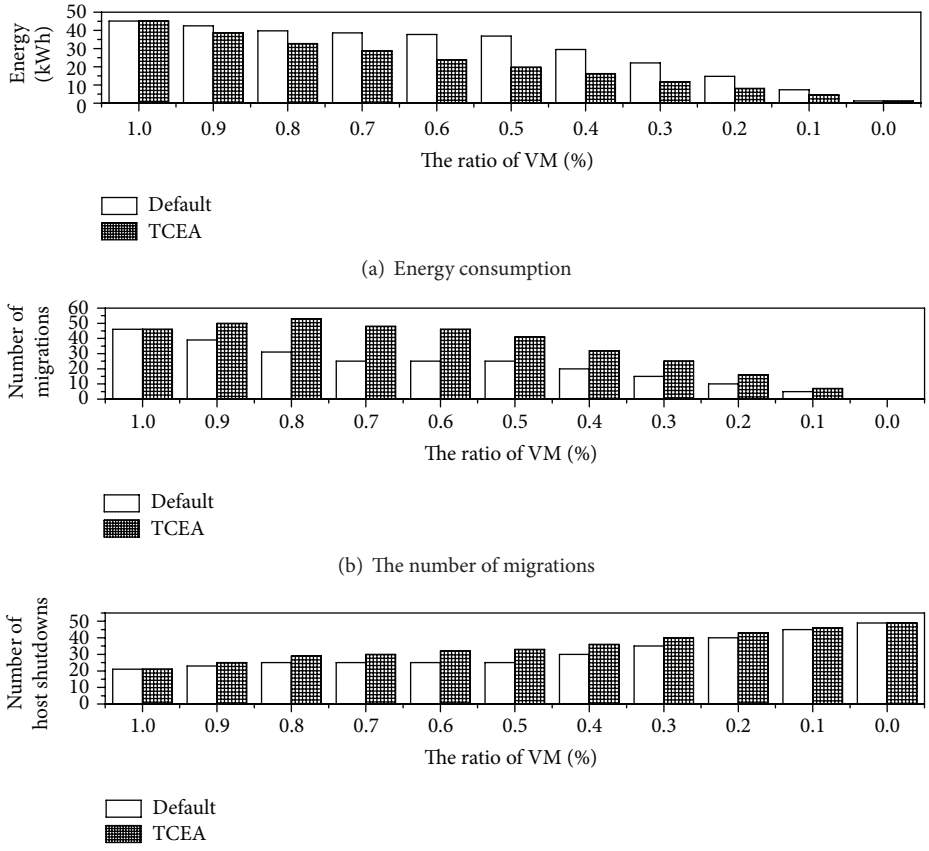


Figure 7: Performance results for lower threshold.

To investigate the respective improvement brought by TCEA's double threshold scheme, we compare the performance of TCEA (double threshold) with the single threshold scheme and default (no threshold and no task classification) setting. In this experiment, we use real task trace logs and artifact task logs for a fixed combination of computation tasks and data-intensive tasks. In Figure 9, "Job" indicates real task traces, *Job_c* indicates only computation-intensive tasks, *Job_d* indicates only data-intensive tasks, and *Job_cd* indicates 50% of computation-intensive tasks and 50% of data-intensive tasks.

As shown in Figure 9, there is no difference for the results with the default setting (no threshold) in terms of energy consumption because a threshold scheme is not applicable. Nevertheless, we leave them for comparison. The double threshold scheme saves 47.6% of energy compared to the default setting. For the single threshold scheme, there is no big difference between 90% and 100% but there are more VM migration operations with 100% of upper threshold, which leads to overheads. Of job categories (*Job*, *Job_c*, *Job_d*, and *Job_cd*), *Job_d* shows a little performance impact with single threshold because it uses relatively less CPU utilization, and *Job_cd* has performance improvement when the single threshold is above 80%. The result for double threshold shows similar phenomenon when the single threshold is used. However, the double threshold scheme further reduces energy consumption by 14.2% compared to the single threshold scheme.

An important requirement for achieving the optimal performance of virtualized cloud environments is to find the appropriate number of VMs per PM. In such an environment, the ratio of PM to VM affects the overall performance. To validate the effect of the ratio of PM to VM, we compare the threshold schemes (default, single, and double). The double scheme achieves the largest energy reduction, followed by the single scheme and by the default scheme as shown in Figure 10. The double threshold scheme saves energy consumption by 11.3% and 27.2% comparing with single and default, respectively. For the number of VM migrations, there are some points where the double threshold scheme exhibits more VM migrations than the single threshold scheme does, but it stabilizes when the ratio of PM to VM is 1 : 9 or more. In addition, the double threshold scheme always outperforms with respect to the number of host shutdowns.

To measure the scalability for the number of PMs and VMs, we increase the number of PMs and VMs from 1 : 2 up to 10 : 20 as shown in Figure 11. As expected, TCEA consumes less energy by 17.9% on average than the default scheme and outnumbers the default scheme in terms of the number of shutdowns. For VM consolidation, TCEA has a higher number of VM migrations. For task scalability, we compare energy consumption by increasing the task log size

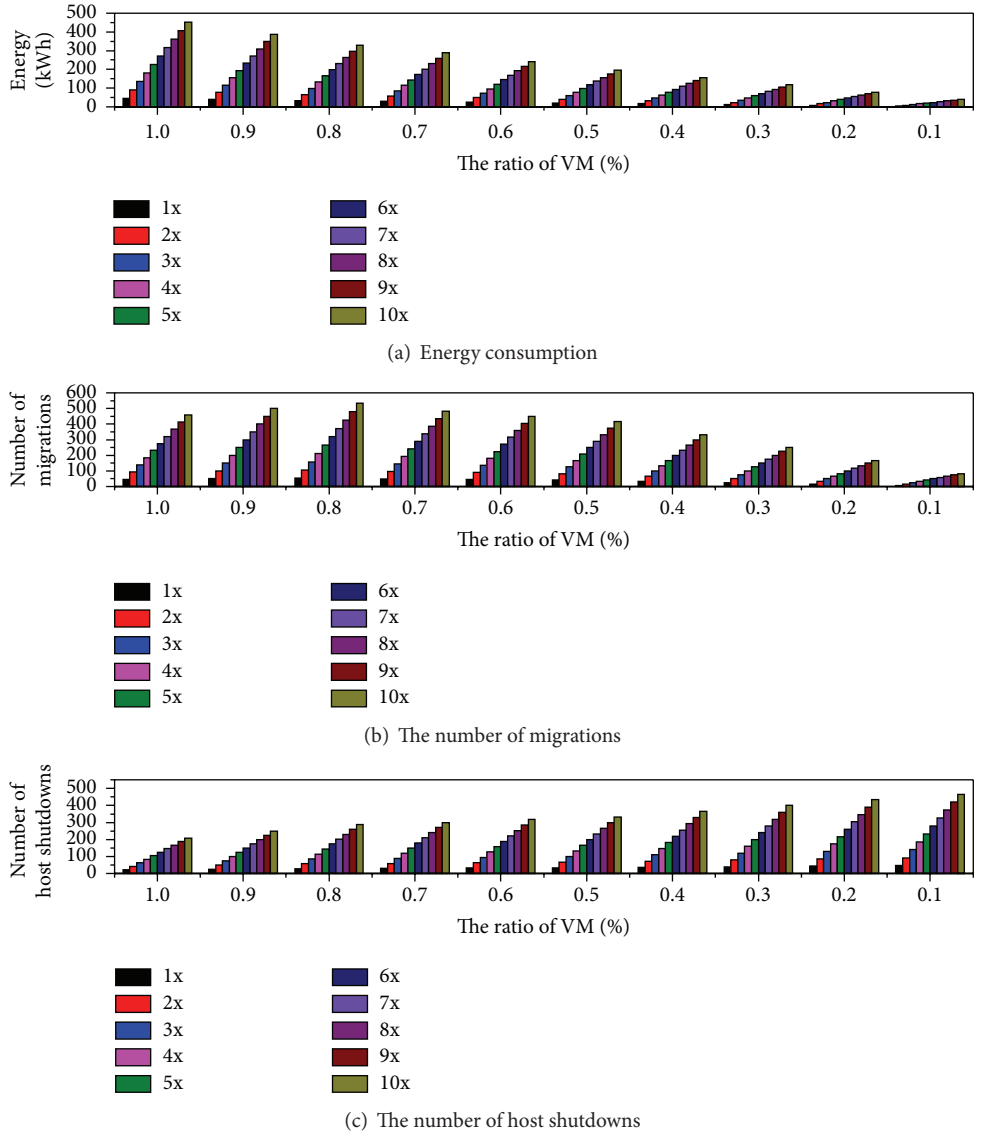


Figure 8: Performance scalability for the number of nodes with lower threshold.

up to 10 times as depicted in Figure 12. Comparing with the default scheme, TCEA consumes less energy by 15.8% on average. Obviously, TCEA has more VM migration and host shutdown operations than the default scheme has for VM consolidation.

5. Related Work

We summarize the related work across three perspectives: resource allocation and scheduling in data centers and clouds, threshold-based schemes with different objectives, and energy savings in data centers. To balance energy consumption and VM utilization, the authors of [10] used a performance to power ratio. It schedules VM migration dynamically and consolidates servers in clouds. They compared their proposed algorithm with three different algorithms including the DVFS algorithm using real trace log files. The authors of [13] proposed a criterion to divide computation-intensive tasks and data-intensive tasks using a communication to computation ratio. The rationale of this task classification is to employ resource allocation methods based on tasks or workflows to improve performance.

In [15], they developed an energy-aware scheduling to reduce total processing time for VMs in a precedence-constrained condition, while maximizing PM's utilization considering communication costs. In [16], they proposed a prediction algorithm for finding overutilized servers and a best-fit algorithm for hosts and VMs. The results show that the algorithms reduce the number of migration operations, rebooting servers, and energy consumption, while achieving SLA guarantee. A separation mechanism of I/O tasks to perform computation-intensive tasks in a batch in virtualized servers to mitigate virtualization overheads is proposed in

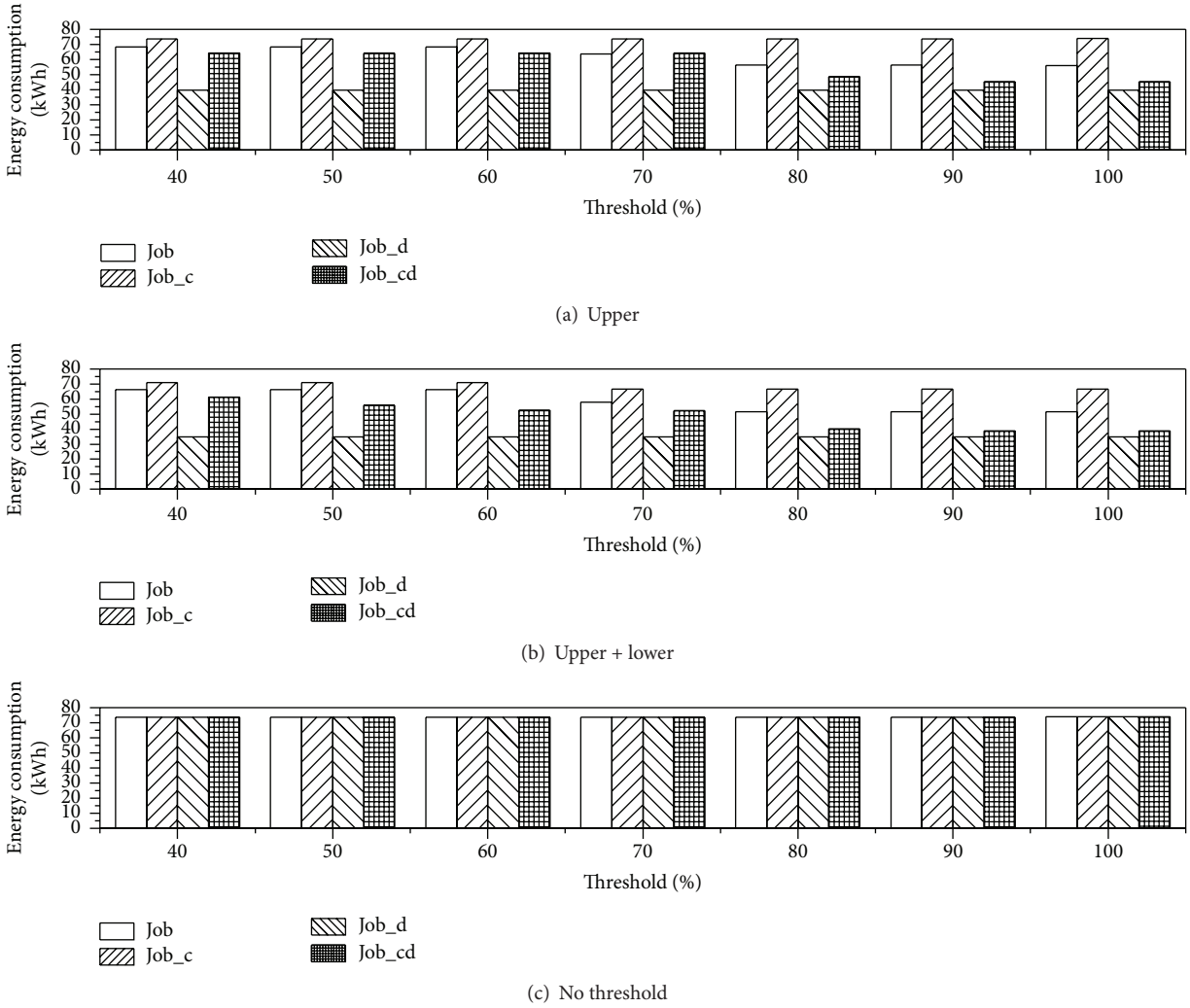
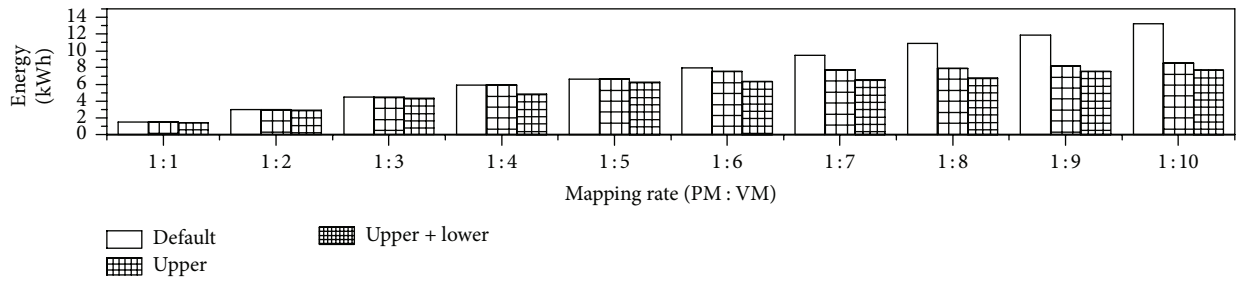


Figure 9: Performance comparison with task types and threshold schemes.

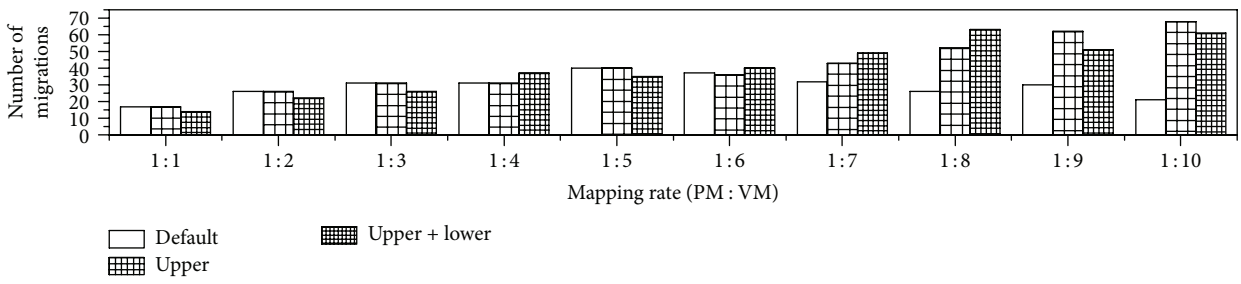
[17]. Because energy consumption and the frequency of SLA violations determine the quality of service [18], in this paper, we balance the tradeoff between energy consumption and SLA violations using the double threshold schemes based on tack classification and none of the abovementioned studies consider the energy saving objectives in the context of task classification.

For data-intensive workflows, where the majority of energy consumption accounts for storing and retrieving data, the authors of [19] consider not using DVFS. Instead, they installed and used an independent node to store data-intensive tasks. They endeavor to reduce energy consumption by minimizing data access and then performed evaluations by increasing the communication to computation ratio. The authors of [20] proposed a VM scheduling algorithm to reduce energy consumption with DVFS. By dynamically adjusting clock frequency and its corresponding voltage, it results in energy reduction in idle and computation stages. In [21], they proposed a scheduling algorithm based on priority and weight with DVFS. It increases servers' resource utilization to reduce energy consumption of the servers. In [22], they used a threshold value to migrate a VM to another host. When a host's utilization is below the threshold value, all the VMs belonging to the host are scheduled to be migrated to other hosts to save idle power consumption. In addition, some VMs are scheduled to be migrated when the host's utilization exceeds a certain threshold value to avoid SLA violations. A service framework that allows monitoring energy consumption and provisioning of VMs to appropriate location in an energy-efficient way is designed in [23].

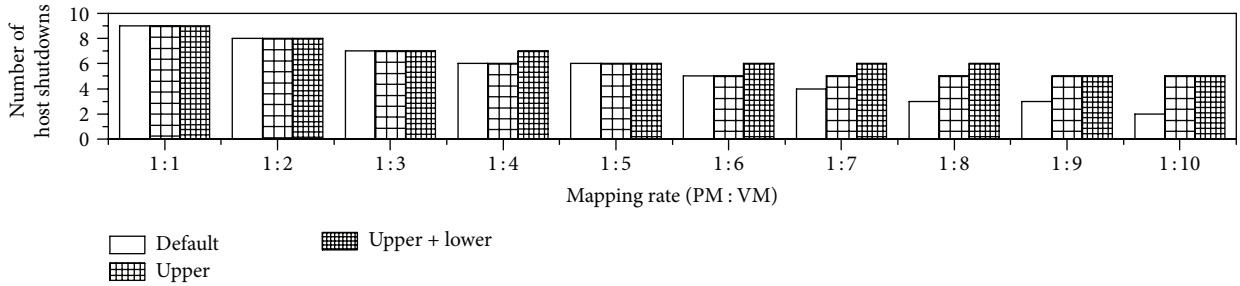
Various CPU consolidation techniques including DVFS, dynamic power shutdown (DPS), and core-level power gating (CPG) are introduced in [24]. The authors of [25] used a threshold value to migrate VMs and consider resource, temperature, and network conditions for optimization. They considered migration time to minimize the number of VMs that are in progress of migration simultaneously. The authors of [26] designed an energy-aware resource allocation heuristic for VMs' initial placement, VM selection policy for migration, and migration policy in virtualized cloud



(a) Energy consumption



(b) The number of migrations



(c) The number of host shutdowns

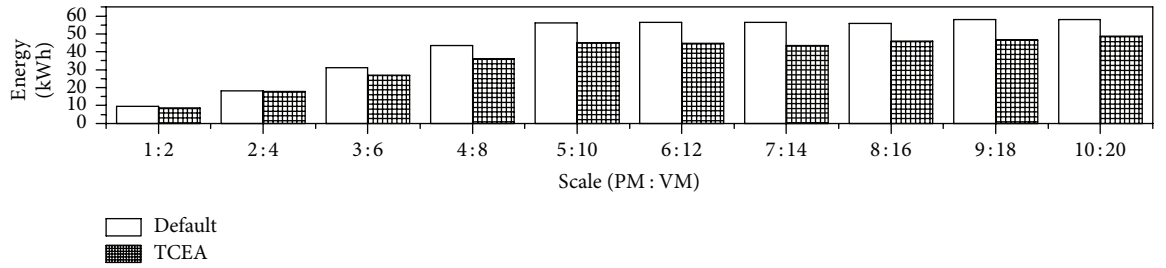
Figure 10: Performance comparison with PM to VM ratio.

computing environments. The authors of [27] developed a resource allocation method at the cloud application level. In the application's perspective, it allocates virtual resources for the application with a threshold-based dynamic resource allocation algorithm to improve resource utilization. In [28], they developed a VM placement algorithm based on the evolutionary game theory. According to their experiments, when the loads of the data center are above 50%, the optimizations are unnecessary.

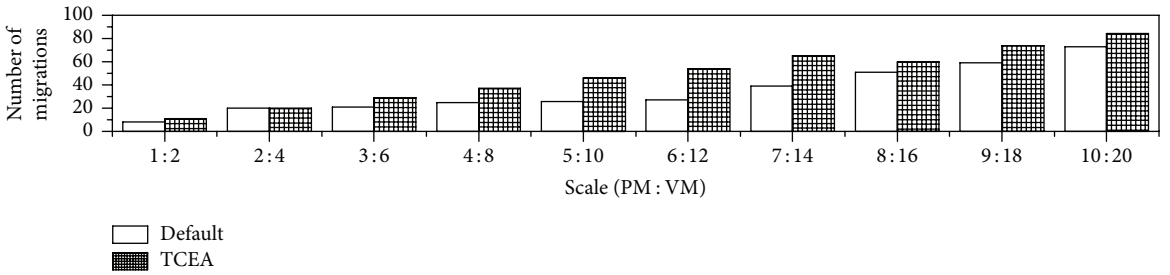
However, the design objective and the implementation methods of these cloud data center schedulers and consolidation algorithms are different from TCEA in terms of the following aspects. First, the target of these cloud data center schedulers is to enforce resource allocation strategy based on fairness or priorities when sharing the resources of large-scale cloud data centers among VMs, while TCEA is aimed at improving both energy consumption and the performance of tasks by dynamically migrating VMs in runtime. Second, we extend a single threshold scheme to further improve the overall performance and energy consumption by incorporating the double threshold scheme and task classification together. Finally, they cannot solve both the maximum utilization problem and the host shutdown problem in an efficient way, while TCEA takes the performance to power ratio into consideration and employs the host shutdown mechanism by migrating VMs on underutilized hosts while maintaining SLA violations.

6. Conclusions

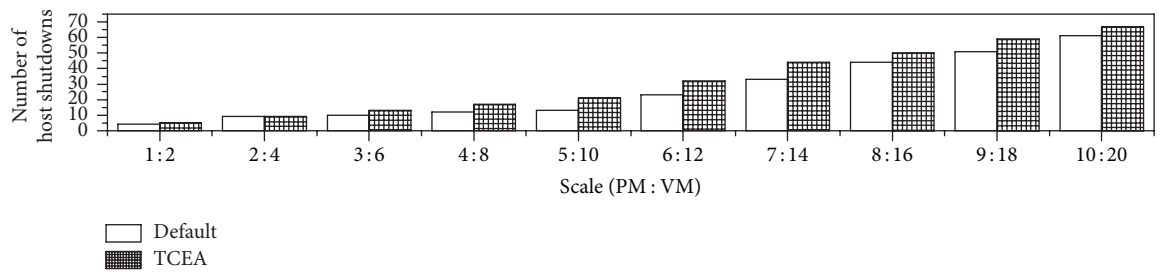
As green IT and its related technologies have received much attention recently, reducing the power consumption of cloud data centers is one of the critical issues to address, thereby reducing the carbon dioxide footprints. In this paper, we propose two consolidation mechanisms for a cloud data center. One is the task consolidation based on task classification (computation-intensive or data-intensive) and the other is the VM consolidation that uses a double threshold scheme (upper and lower). We optimize energy consumption in a virtualized data center not by maximizing resource utilization but by balancing resource utilization of hosts with migrating appropriate VMs. We prove that our task classification based energy-aware consolidation algorithm (TCEA) achieves significant energy reduction without incurring predefined SLA violations.



(a) Energy consumption



(b) The number of migrations



(c) The number of host shutdowns

Figure 11: Performance comparison with scalability.

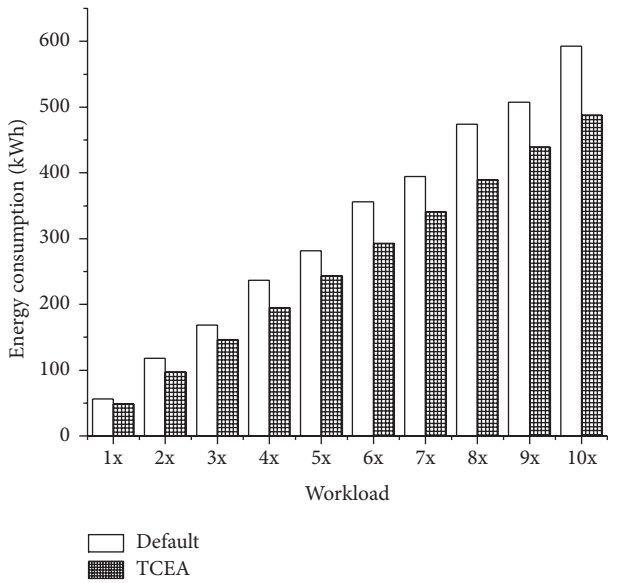


Figure 12: Scalability for the number of tasks.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2015R1C1A2A01054813).

References

[1] W. Ai, K. Li, S. Lan et al., "On elasticity measurement in cloud computing," *Scientific Programming*, vol. 2016, Article ID 7519507, 13 pages, 2016.

[2] J. Lim, T. Suh, J. Gil, and H. Yu, "Scalable and leaderless Byzantine consensus in cloud computing environments," *Information Systems Frontiers*, vol. 16, no. 1, pp. 19–34, 2014.

[3] S. K. Choi, K. S. Chung, and H. Yu, "Fault tolerance and QoS scheduling using CAN in mobile social cloud computing," *Cluster Computing*, vol. 17, no. 3, pp. 911–926, 2014.

[4] M. Armbrust, A. Fox, R. Griffith et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[5] Y. Wen, X. Zhu, J. J. P. C. Rodrigues, and C. W. Chen, "Cloud mobile media: reflections and outlook," *IEEE Transactions on Multimedia*, vol. 16, no. 4, pp. 885–902, 2014.

[6] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: a survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2015.

[7] N. Boumkheld, M. Ghogho, and M. E. Koutbi, "Energy consumption scheduling in a smart grid including uding renewable energy," *Journal of Information Processing Systems*, vol. 11, no. 1, pp. 116–124, 2015.

[8] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[9] I. Habib, "Virtualization with KVM," *Linux Journal*, vol. 2008, no. 166, article 8, 2008.

[10] X. Ruan and H. Chen, "Performance-to-power ratio aware Virtual Machine (VM) allocation in energy-efficient clouds," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '15)*, pp. 264–273, Chicago, Ill, USA, September 2015.

[11] D. Sood, H. Kour, and S. Kumar, "Survey of computing technologies: distributed, utility, cluster, grid and cloud computing," *Journal of Network Communications and Emerging Technologies*, vol. 6, no. 5, pp. 99–102, 2016.

[12] Y. Gao, H. Guan, Z. Qi, B. Wang, and L. Liu, "Quality of service aware power management for virtualized data centers," *Journal of Systems Architecture*, vol. 59, no. 4-5, pp. 245–259, 2013.

[13] W. Guo, W. Sun, W. Hu, and Y. Jin, "Resource allocation strategies for data-intensive workflow-based applications in optical grids," in *Proceedings of the 10th IEEE Singapore International Conference on Communications Systems (ICCS '06)*, pp. 1–5, IEEE, Singapore, November 2006.

[14] O. Shai, E. Shmueli, and D. G. Feitelson, "Heuristics for resource matching in Intel's compute farm," in *Job Scheduling Strategies for Parallel Processing*, vol. 8429, pp. 116–135, Springer, 2013.

[15] V. Ebrahimirad, M. Goudarzi, and A. Rajabi, "Energy-aware scheduling for precedence-constrained parallel virtual machines in virtualized data centers," *Journal of Grid Computing*, vol. 13, no. 2, pp. 233–253, 2015.

[16] J. Huang, K. Wu, and M. Moh, "Dynamic Virtual Machine migration algorithms using enhanced energy consumption model for green cloud data centers," in *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS '14)*, pp. 902–910, Bologna, Italy, July 2014.

[17] P. Xiao, Z. Hu, D. Liu, X. Zhang, and X. Qu, "Energy-efficiency enhanced virtual machine scheduling policy for mixed workloads in cloud environments," *Computers & Electrical Engineering*, vol. 40, no. 5, pp. 1650–1665, 2014.

[18] A. Paya and D. C. Marinescu, "Energy-aware load balancing policies for the cloud ecosystem," in *Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW '14)*, pp. 823–832, IEEE, Phoenix, Ariz, USA, May 2014.

[19] P. Xiao, Z.-G. Hu, and Y.-P. Zhang, "An energy-aware heuristic scheduling for data-intensive workflows in virtualized datacenters," *Journal of Computer Science and Technology*, vol. 28, no. 6, pp. 948–961, 2013.

[20] G. von Laszewski, L. Wang, A. J. Younge, and X. He, "Power-aware scheduling of virtual machines in DVFS-enabled clusters," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops (CLUSTER '09)*, pp. 1–10, IEEE, New Orleans, La, USA, September 2009.

[21] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Generation Computer Systems*, vol. 37, pp. 141–147, 2014.

[22] L. Luo, W. Wu, W. Tsai, D. Di, and F. Zhang, "Simulation of power consumption of cloud data centers," *Simulation Modelling Practice and Theory*, vol. 39, pp. 152–171, 2013.

[23] G. Katsaros, J. Subirats, J. O. Fitó, J. Guitart, P. Gilet, and D. Espling, "A service framework for energy-aware monitoring and VM management in Clouds," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2077–2091, 2013.

[24] I. Hwang, T. Kam, and M. Pedram, "A study of the effectiveness of CPU consolidation in a virtualized multi-core server system," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '12)*, pp. 339–344, Redondo Beach, Calif, USA, August 2012.

[25] K. Maurya and R. Sinha, "Energy conscious dynamic provisioning of virtual machines using adaptive migration thresholds in cloud data center," *International Journal of Computer Science and Mobil Computing*, vol. 2, no. 3, pp. 74–82, 2013.

[26] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[27] W. Lin, J. Z. Wang, C. Liang, and D. Qi, "A threshold-based dynamic resource allocation scheme for cloud computing," *Procedia Engineering*, vol. 23, pp. 695–703, 2011.

[28] Z. Xiao, J. Jiang, Y. Zhu, Z. Ming, S. Zhong, and S. Cai, "A solution of dynamic VMs placement problem for energy consumption optimization based on evolutionary game theory," *Journal of Systems and Software*, vol. 101, pp. 260–272, 2015.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 3101658, 10 pages
http://dx.doi.org/10.1155/2016/3101658

Research Article

Optimized Virtual Machine Placement with Traffic-Aware Balancing in Data Center Networks

Tao Chen, Xiaofeng Gao, and Guihai Chen

Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Correspondence should be addressed to Xiaofeng Gao; gao-xf@cs.sjtu.edu.cn

Received 26 February 2016; Revised 22 July 2016; Accepted 1 August 2016

Academic Editor: Ligang He

Virtualization has been an efficient method to fully utilize computing resources such as servers. The way of placing virtual machines (VMs) among a large pool of servers greatly affects the performance of data center networks (DCNs). As network resources have become a main bottleneck of the performance of DCNs, we concentrate on VM placement with Traffic-Aware Balancing to evenly utilize the links in DCNs. In this paper, we first proposed a Virtual Machine Placement Problem with Traffic-Aware Balancing (VMPPTB) and then proved it to be NP-hard and designed a Longest Processing Time Based Placement algorithm (LPTBP algorithm) to solve it. To take advantage of the communication locality, we proposed Locality-Aware Virtual Machine Placement Problem with Traffic-Aware Balancing (LVMPPTB), which is a multiobjective optimization problem of simultaneously minimizing the maximum number of VM partitions of requests and minimizing the maximum bandwidth occupancy on uplinks of Top of Rack (ToR) switches. We also proved it to be NP-hard and designed a heuristic algorithm (Least-Load First Based Placement algorithm, LLBP algorithm) to solve it. Through extensive simulations, the proposed heuristic algorithm is proven to significantly balance the bandwidth occupancy on uplinks of ToR switches, while keeping the number of VM partitions of each request small enough.

1. Introduction

As virtualization technology [1] becomes the mainstream way to multiplex various physical resources in modern cloud data centers, the effective and efficient placement of virtual machines (VMs) becomes an important issue. Mechanisms such as VMware Capacity Planner [2] and Novell PlateSpin Recon [3] consolidate VMs such that the consumption of CPU, memory, and power are optimized. Owing to the increasing deployment of communication-intensive applications like MapReduce [4], the data center network (DCN) is becoming the bottleneck of applications performance and scalability. Thus, those mechanisms without considering network resources are not feasible in cloud data centers.

Three-layer tree-like architecture is prevalently used in modern data centers [5], as shown in Figure 1. This kind of architecture, however, inherently suffers scalability issue due to the fact that links connected to the core layer switches usually transfer more traffic from lower layers. Physical machines (PMs) connected to the same Top of Rack (ToR) switch can communicate at full line speed, and the traffic between PMs connected to different ToR switches has to traverse across links of the core layer, which is often the bottleneck of data center networks (DCNs). In this paper, we placed VMs on PMs by effectively balancing traffic in the core layer of DCNs to minimize the maximum bandwidth occupancy on uplinks of the Top of Rack (ToR) switches.

The issue on scalability of DCNs has attracted great attention from academia recently. Several recent works address this issue by designing new DCN architectures, aiming at maximizing the network bisection bandwidth [6–8] and reducing the overall oversubscription ratio of the DCN. Other papers [9–11] address this issue by optimizing VM placements on PMs with different optimization goals. The bisection bandwidth can be improved; however, the available bandwidth in the core layer cannot be fully utilized. The number of highly utilized links in the core layer never exceeds 25% [12], which means a great number of links in the core layer are underutilized. Our proposed VM placement with Traffic-Aware Balancing can evenly spread traffic across links

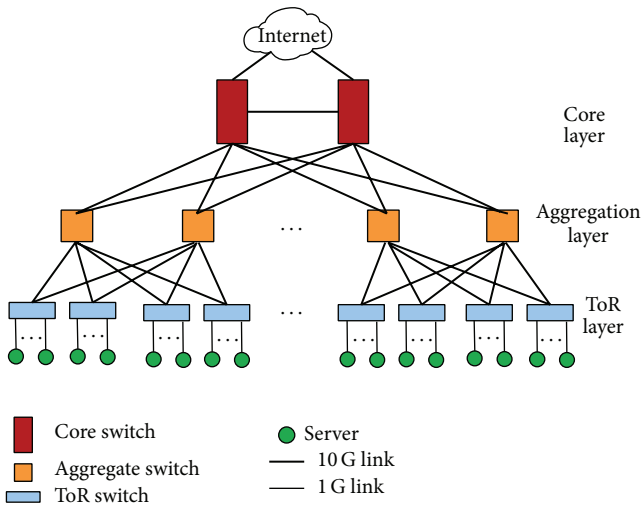


Figure 1: An example of three-layer tree-like architecture.

of the core layer to utilize the high bisection bandwidth. Since link utilizations in the core/aggregation layers are higher than that in the ToR layer and a significant fraction of the core links appear as hotspots persistently [12, 13], we only consider the traffic across the core layer of DCNs, and the traffic between VMs of the same request under the same ToR switch does not contribute any traffic towards the core layer.

We formally defined our Virtual Machine Placement Problem with Traffic Balancing (VMPPTB) as an optimization problem, which minimizes the maximum bandwidth occupancy on uplinks of each ToR switch. We proved its NP-hardness by reducing from the Multiprocessor Scheduling Problem [14] and designed a Longest Processing Time Based Placement algorithm (LPTBP algorithm) to solve it. The LPTBP algorithm provides an optimum solution to the VMPPTB problem; however, the generated placement schema tends to evenly place VMs of each request under every ToR switch. As the fact that VMs of a tenant's request only communicate with other VMs within the same request (communication locality property), we should reduce the number of VM partitions of each request at the same time (i.e., placing VMs of the same request on as few PMs as possible). We further proposed Locality-Aware Virtual Machine Placement Problem with Traffic Balancing (LVMPPTB), which aims to minimize the maximum number of VM partitions of each request and the maximum bandwidth occupancy on uplinks of ToR switches simultaneously. We also proved it to be NP-hard by reducing from VMPPTB and designed a Least-Load First Based Placement algorithm (LLBP algorithm) to solve it.

We summarize our contribution as follows:

(i) We formally formulated the Virtual Machine Placement Problem with Traffic Balancing (VMPPTB), proved its computation complexity, and designed a Longest Processing Time Based Placement algorithm (LPTBP algorithm).

(ii) To take advantage of the communication locality property, we further proposed the Locality-Aware Virtual Machine Placement Problem with Traffic Balancing (LVMPPTB), proved its NP-hardness, and designed a Least-Load First Based Placement algorithm (LLBP algorithm).

(iii) We designed a Greedy Based Placement algorithm (GBP algorithm) as the expected baseline, and took the LPTBP algorithm as the optimum solution. We evaluated the performance of LLBP through extensive simulations, compared with GBP algorithm and LPTBP algorithm.

The rest of this paper is organized as follows. We briefly present related work in Section 2. Problem formulation and computation complexity proofs are presented in Section 3. In Section 4, we describe the LPTBP and LLBP algorithms. We evaluate our algorithms in Section 5. Finally, we make a conclusion in Section 6.

2. Related Work

In [9], the authors addressed the network scalability issue by using traffic-aware virtual machine (VM) placement. They defined the placement problem as an optimization problem to minimize the communication costs of all the VMs, where communication cost is defined as the hops between each VM pair. They assumed that the traffic matrices between virtual machines are known in advance. The generated placement scheme can reduce the communication distance between VMs with large traffic and reduce aggregated traffic into the higher level of the data center network (DCN) architecture.

In [10], the authors proposed jointly optimizing virtual machine placement and route selecting. They strived to minimize the averaged congestion rate of every link in DCNs. Their placement strategy performs better in topologies with rich connectivity and path diversity. Although the averaged congestion rate is minimized, the traffic in DCNs may not be significantly balanced. The traffic matrix is also assumed to be known in advance.

In [11], the authors presented a Min-Cut Ratio-Aware VM Placement (MCRVMP) problem. They tried to minimize the maximum ratio of the demand and capacity across all cuts in the network, where the cut is defined as a set of links that partition the hosts into two disjoint connected components, the capacity is the sum capacity of the links, and the demand is the total traffic from either side of the hosts. In this way, each network cut may have spare capacity to absorb unpredicted traffic bursts. This work is only used in medium sized data centers, and traffic rates are assumed to be known in advance.

Knowing the traffic matrix in advance is a very strong assumption. In [15], the authors proposed to adopt the product traffic pattern model to characterize the traffic rates between VMs, where the traffic rate is defined as the product of activity levels of two communicating VMs. Similar to the objective of [9], they tended to place more active VMs into physical machines (PMs) with less communication cost. The proposed product traffic pattern is unrealistic to some degree. The generated placement scheme may result in hotspots, as VMs with higher activity levels are placed in hosts connected

to the same switch; the uplinks of that switch may become hotspots.

In [16], the authors formulated the placement problem as an optimization problem to minimize the total cost caused by network traffic and utilization of PMs. When the number of PMs is fixed, the authors proposed three different traffic cost functions to solve the optimization problem. The data center topology, however, is not taken into consideration when designing the traffic cost functions.

3. Problem Formulation

In this section, we define the VM placement problem based on tree-like architectures such as fat-tree [6] and VL2 [7]. Data centers usually leverage three-layer tree-like architectures, in which physical machines (PMs) are directly connected with Top of Rack (ToR) switches, ToR switches are connected with aggregation switches, and aggregation switches are further connected with core switches. The tree-like topology, however, is often oversubscribed, due to the fact that a higher level link has to carry traffic from several lower level links. As the higher level links are often the bottlenecks and hotspots [12], we explore to balance traffic on links of the core layer of the data center networks (DCNs).

In the VM placement problem, the traffic between VMs connected to the same ToR switch does not transfer across the core switch layer of DCNs. We only need to consider traffic on the uplinks of ToR switches, since the traffic can be dynamically load balanced among the aggregation layer and the core layer using load balancing mechanisms like Valiant Load Balancing [6]. By properly placing VMs of multiple requests, we can better evenly and effectively utilize every link of core layer of the DCNs and thus minimize the maximum bandwidth occupancy on uplinks of ToR switches.

3.1. VMPPTB Problem. We define the VM placement problem in the scenario where the DCN contains n ToR switches, represented as a set $\mathbb{T} = \{T_1, T_2, \ldots, T_n\}$. For each ToR switch, we can place up to c VMs in one PM connected with it. Let L_k be the uplink of the kth ToR switch, and let B_{L_k} be the accumulated bandwidth occupancy on uplink L_i. Suppose there are m requests $\mathbb{R} = \{R_1, R_2, \ldots, R_m\}$ from different tenants in the cloud data center, and the corresponding numbers of requested VM are $\mathbb{S} = \{s_1, s_2, \ldots, s_m \mid \forall i,\ s_i > c\}$. If $s_i \leqslant c$, the requested VMs could be placed under the same ToR switch, and when $s_i > c$, the requested VMs must be partitioned into several PMs. The bandwidth requirement of VM j of request R_i is denoted as B_{ij}. If VM j was placed under a ToR switch T_k, it would contribute B_{ij} bandwidth occupancy on uplink L_k. We should properly place all the VMs of requests under n ToR switches to minimize the maximum bandwidth occupancy on the uplinks of ToR switches with the constraint that each VM should be placed under some ToR switch and the number of VMs placed under one ToR switch should be no more than c.

Let D_{ij}^k be a binary indicator of whether VM j of request R_i is placed under ToR T_k. Consider

$$D_{ij}^k = \begin{cases} 1, & \text{VM } j \text{ of request } R_i \text{ is placed under } T_k \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The descriptions of the symbols used in this paper are summarized in Notations.

We formally define the Virtual Machine Placement Problem with Traffic Balancing (VMPPTB) as follows:

$$\min \quad \max_{k \in [1,n]} B_{L_k} \tag{2}$$

$$\text{s.t.} \quad \sum_{j=1}^{s_i} \sum_{k=1}^{n} D_{ij}^k = s_i, \quad \forall i \in \{1, 2, \ldots, m\} \tag{3}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{s_i} D_{ij}^k \leqslant c, \quad \forall k \in \{1, 2, \ldots, n\} \tag{4}$$

$$B_{L_k} = \sum_{i=1}^{m} \sum_{j=1}^{s_i} D_{ij}^k \cdot B_{ij}, \quad \forall k \in \{1, 2, \ldots, n\}. \tag{5}$$

The objective function (2) minimizes the maximum bandwidth occupancy on uplinks of ToR switches. Constraint (3) ensures that each VM of requests is placed under some ToR switch. Constraint (4) guarantees that the number of VMs under every ToR switch is not more than the capacity of one PM. Constraint (5) updates bandwidth occupancy on uplink L_k when we place a VM under ToR switch T_k.

We prove VMPPTB is a NP-hard problem.

Theorem 1. *For the Virtual Machine Placement Problem with Traffic Balancing (VMPPTB) defined above, finding its optimal solution is NP-hard.*

Proof. This can be proven by a reduction from the Multiprocessor Scheduling Problem (MSP). Given a set $\mathbb{J}$ of independent jobs and a number of processors M and given that job J_i has length I_i, what is the minimum possible time required to schedule all jobs in $\mathbb{J}$ on M processors such that none overlap? The MSP is known to be a NP-hard problem [14].

For example, we make the set $\mathbb{J}$ as a request of some tenant, job J_i as VM_j of request R_m, the length I_i of job J_i as bandwidth requirement B_{mj} of VM_j of request R_m, and M processors as n ToR switches. It turns out to be an instance of VMPPTB, except that the number of VMs placed under a ToR switch is limited to c in VMPPTB. However, we can set c extremely large such that the constraint holds in any placement schema. In this way, the MSP can be reduced to the VMPPTB. Thus, we prove the VMPPTB is a NP-hard problem. □

Since we have proven that VMPPTB is NP-hard by reducing from MSP, we can solve VMPPTB by approximation algorithms designed to solve MSP. A simple but classical algorithm called Longest Processing Time (LPT) algorithm

can achieve an upper bound of $(4/3 - 1/3M)$OPT [17]. It is feasible to design a Longest Processing Time Based Placement (LPTBP) algorithm to solve VMPPTB. We first sort bandwidth requirements of VMs of all requests in nonincreasing order. Then we place the VM with the maximum bandwidth requirement under the ToR switch, of which the uplink has minimum bandwidth occupancy. We repeat the process until VMs of all requests are placed on PMs.

3.2. LVMPPTB Problem. The LPTBP algorithm can generate an approximate optimal solution to VMPPTB; however, it cannot take the communication locality of VMs of a request into consideration (i.e., VMs of a request only communicate with other VMs within the same request). Thus, if all VMs of a request are placed under as few ToR switches as possible, it could essentially reduce the traffic forward to the core layer and further mitigate hotspots in the core layer.

Let $\mathbb{T}_{R_i} = \{T_k \mid D_{ij}^k = 1,\ \forall j \in \{1, 2, \ldots, s_i\}\}$ be a subset of $\mathbb{T}$, which contains the ToR switches under which the VMs of request R_i are placed. $|\mathbb{T}_{R_i}|$ denotes the number of VM partitions of request R_i, which should be minimized to effectively take advantage of the communication locality property. The multiobjective optimization problem named Locality-Aware Virtual Machine Placement Problem with Traffic Balancing (LVMPPTB) aims to minimize the maximum bandwidth occupancy on the uplinks of ToR switches and minimize the maximum number of VM partitions of all the requests simultaneously, which is formally defined as follows:

$$
\begin{aligned}
&\min \quad \max_{k\in[1,n]} B_{L_k} \\
&\min \quad \max_{i\in[1,m]} \left|\mathbb{T}_{R_i}\right| \\
&\text{s.t.} \quad \sum_{j=1}^{s_i}\sum_{k=1}^{n} D_{ij}^k = s_i, \quad \forall i \in \{1, 2, \ldots, m\} \\
&\qquad \sum_{i=1}^{m}\sum_{j=1}^{s_i} D_{ij}^k \leqslant c, \quad \forall k \in \{1, 2, \ldots, n\} \\
&\qquad B_{L_k} = \sum_{i=1}^{m}\sum_{j=1}^{s_i} D_{ij}^k \cdot B_{ij}, \quad \forall k \in \{1, 2, \ldots, n\} \\
&\qquad \mathbb{T}_{R_i} = \left\{T_k \mid D_{ij}^k = 1\right\}, \quad \forall j \in \left\{1, 2, \ldots, s_i\right\}.
\end{aligned}
\tag{6}
$$

We also prove LVMPPTB is a NP-hard problem.

Theorem 2. *For the Locaility-Aware Virtual Machine Placement Problem with Traffic Balancing (LVMPPTB), finding its optimal solution is NP-hard.*

Proof. Since we have proven that the VMPPTB is a NP-hard problem, we can prove LVMPPTB's NP-hardness by reducing it from the VMPPTB. A special instance of LVMPPTB is that there is only one request in the DCN. Therefore, the number of VM partitions of this request equals the number of VMs of this request divided by the maximum number of VMs placed under a ToR switch. It turns out to be the same with the VMPPTB. If we can find an optimal solution to the VMPPTB, we can also find an optimal solution to this special instance of LVMPPTB and vice versa. As the VMPPTB is NP-hard, the LVMPPTB's NP-hardness is proven. □

4. Algorithms

We have proven that the Virtual Machine Placement Problem with Traffic Balancing (VMPPTB) is a NP-hard problem. As the VMPPTB is the reduction from the Multiprocessor Scheduling Problem, we designed a Longest Processing Time Based Placement (LPTBP) algorithm to solve the VMPPTB, as shown in Algorithm 1.

The LPTBP algorithm aims to achieve balanced bandwidth occupancy of uplinks of all ToR switches by placing the VM with maximum bandwidth requirement under the ToR switch with minimum bandwidth occupancy every time. First, $\{B_{ij}\}$ are put into a two-dimensional array $\mathbb{V}$. $\{B_{L_k}\}$ and $\{C_k\}$ are initially $\emptyset$. For the VM with maximum bandwidth requirement, the LPTBP algorithm selects the ToR switch with minimum bandwidth occupancy, places the VM under it, and then updates $\mathbb{V}$, $\mathbb{B}$, and $\mathbb{C}$. The LPTBP algorithm repeats the process until no VMs can be placed under any ToR switches and outputs a virtual machine placement schema.

Though the LPTBP algorithm has an approximation ratio $(4/3 - 1/3M)$OPT [17], the output placement schema does not take advantage of the communication locality property. Hence, we propose a heuristic algorithm named Least-Load First Based Placement (LLBP) to solve the Locality-Aware Virtual Machine Placement Problem with Traffic Balancing (LVMPPTB), as shown in Algorithm 2.

The LLBP algorithm places the requests in the nonincreasing order of their numbers of VMs. For each request, LLBP tries to find a minimum *empty* ToR switch set that can hold all its VMs. The ToR switch(es) in an empty ToR switch set is(are) fully available and connected to none of VMs. If the *empty* ToR switch set exists, LLBP places all VMs of the request in the nonincreasing order of bandwidth requirement under the empty ToR switch set in the least-load first way (placing the VM with maximum bandwidth requirement under the ToR switch with minimum bandwidth occupancy). If no such *empty* ToR switch set exists for a request, LLBP tries to find a minimum ToR switch set to hold its VMs and also place the VMs under the ToR switch set in the least-load first way.

LLBP first places VMs of the request with maximum number of VMs, and an *empty* ToR switch set exists to meet the communication locality property, and the number of VM partitions would be small. LLBP repeats the process until no such *empty* ToR switch set exists. At this time, for these requests with smaller numbers of VMs, LLBP tries to find some ToR switch sets to hold their VMs, which are placed under several ToR switches, and the numbers of VM partitions would rise. The worst case is that each VM of a request is placed under one ToR switch, in which the communication locality property is hard to meet. However, as the number of VMs is smaller, the number of VM partitions would be smaller and the overhead of communication between these

Input: $\{B_{ij}\}$: Set of bandwidth requirements of VMs of all requests;
$\{B_{L_k}\}$: Set of cumulative bandwidth occupancy of the uplinks of all ToR switches;
$\{C_k\}$: Set of cumulative numbers of VMs under ToR switches
B_{L_c}: Maximum bandwidth capacity of a uplink of ToR switch;
c: Maximum number of VMs under a ToR switch.
Output: Virtual Machine Placement Schema
(1) $\mathbb{V} \leftarrow \{B_{ij}\}, i \in \{1, 2, \ldots, m\}, j \in \{1, 2, \ldots, s_i\}$
(2) $\mathbb{B} \leftarrow \{B_{L_k}\}, k \in \{1, 2, \ldots, n\}$
(3) $\mathbb{C} \leftarrow \{C_k\}, k \in \{1, 2, \ldots, n\}$
(4) $\mathbb{B} = \emptyset$
(5) $\mathbb{C} = \emptyset$
(6) **for** $v \leftarrow 1$ to $\sum_{i=1}^{m} s_i$ **do**
(7) $u \leftarrow \arg\min_k B[k]$
(8) $(p, q) \leftarrow \arg\max_{i,j} V[i][j]$
(9) **while** $(B[u] + V[p][q] \leqslant B_{L_c}$ && $C[u] \leqslant c)$ **do**
(10) place VM $q + 1$ of request R_{p+1} under ToR switch T_{u+1}
(11) $B[u] \leftarrow B[u] + V[p][q]$
(12) $C[u] \leftarrow C[u] + 1$
(13) $0 \leftarrow V[p][q]$
(14) **end while**
(15) **end for**
(16) **return** Virtual Machine Placement Schema

Algorithm 1: Longest Processing Time Based Placement (LPTBP).

Input: $\mathbb{S} = \{s_i\}$: Set of numbers of VMs of requests
$\{B_{ij}\}$: Set of bandwidth requirements of VMs of all requests;
$\{B_{L_k}\}$: Set of cumulative bandwidth occupancy of the uplinks of all ToR switches;
$\{C_k\}$: Set of cumulative numbers of VMs under ToR switches
B_{L_c}: Maximum bandwidth capacity of a uplink of ToR switch;
c: Maximum number of VMs under a ToR switch.
Output: Virtual Machine Placement Schema
(1) $\mathbb{S} \leftarrow \{s_i\}, i \in \{1, 2, \ldots, m\}$
(2) $\mathbb{V} \leftarrow \{B_{ij}\}, i \in \{1, 2, \ldots, m\}, j \in \{1, 2, \ldots, s_i\}$
(3) $\mathbb{B} \leftarrow \{B_{L_k}\}, k \in \{1, 2, \ldots, n\}$
(4) $\mathbb{C} \leftarrow \{C_k\}, k \in \{1, 2, \ldots, n\}$
(5) $\mathbb{B} = \emptyset$
(6) $\mathbb{C} = \emptyset$
(7) **for** $i \leftarrow 1$ to m **do**
(8) $u \leftarrow \arg\max_i S[i]$
(9) find a minimum *empty* ToR switch set $\mathbb{T}_{R_{u+1}}$ to hold all VMs of request R_{u+1}, $|\mathbb{T}_{R_{u+1}}| = \lceil s_{u+1}/c \rceil$
(10) **if** $\mathbb{T}_{R_{u+1}} = \emptyset$ **then**
(11) find a minimum ToR switch set $\mathbb{T}'_{R_{u+1}}$ to hold all VMs of request R_{u+1}
(12) **end if**
(13) place all VMs of request R_{u+1} in the non-increasing order of bandwidth requirement under $\mathbb{T}_{R_{u+1}}$ or $\mathbb{T}'_{R_{u+1}}$ in the least-load first way under constraints B_{L_k}, c
(14) update $\mathbb{B}$, $\mathbb{C}$ by $V[u][0], V[u][1], \ldots, V[u][s_{u+1} - 1], s_{u+1}$
(15) **end for**

Algorithm 2: Least-Load First Based Placement (LLBP).

VMs is smaller. The VMs of all requests are placed in a least-load first way, and the bandwidth occupancy on uplinks of ToR switches can be balanced significantly. Therefore, LLBP achieves a better trade-off between the number of VM partitions of requests and bandwidth occupancy of uplinks of the ToR switches.

We also design a Greedy Based Placement (GBP) algorithm, as shown in Algorithm 3. The GBP algorithm places VMs of all requests sequentially under ToR switches in the ToR switch order, which means that each VM of a request is placed as close as possible to other VMs of the same request.

Input: $\{B_{ij}\}$: Set of bandwidth requirements of VMs of all requests;
$\{B_{L_k}\}$: Set of cumulative bandwidth occupancy of the uplinks of all ToR switches;
$\{C_k\}$: Set of cumulative numbers of VMs under ToR switches
B_{L_c}: Maximum bandwidth capacity of a uplink of ToR switch;
c: Maximum number of VMs under a ToR switch.
Output: Virtual Machine Placement Schema
(1) $\mathbb{V} \leftarrow \{B_{ij}\}, i \in \{1, 2, \ldots, m\}, j \in \{1, 2, \ldots, s_i\}$
(2) $\mathbb{B} \leftarrow \{B_{L_k}\}, k \in \{1, 2, \ldots, n\}$
(3) $\mathbb{C} \leftarrow \{C_k\}, k \in \{1, 2, \ldots, n\}$
(4) $\mathbb{B} = \emptyset$
(5) $\mathbb{C} = \emptyset$
(6) **for** $i \leftarrow 1$ to m **do**
(7) **for** $j \leftarrow 1$ to s_i **do**
(8) **for** $k \leftarrow 1$ to n **do**
(9) **while** $(B[k-1] + V[i-1][j-1] \leqslant B_{L_c}$ && $C[k-1] \leqslant c)$ **do**
(10) place VM j of request R_i under ToR switches in the ToR switch order
(11) **end while**
(12) **end for**
(13) **end for**
(14) **end for**

Algorithm 3: Greedy Based Placement (GBP).

We illustrate the three algorithms by an example, as shown in Figure 2. There are 3 requests, R_1, R_2, and R_3, which are placed under 5 ToR switches, T_1, T_2, T_3, T_4, and T_5. A ToR switch can be connected with up to three VMs. The numbers of VMs of 3 requests are 4, 5, and 6, respectively. The sets of bandwidth requirements of VMs of requests are $\{25, 125, 225, 325\}$, $\{50, 150, 250, 350, 450\}$, and $\{100, 200, 300, 400, 500, 600\}$, respectively. The results of the three algorithms are shown in Figures 2(a)–2(c). The output placement schema of LTPBP algorithm achieves the best balance on the bandwidth occupancy of uplinks of ToR switches (775~825). However, the numbers of VM partitions of requests are all more than 3. The output placement schema of LLFBP algorithm achieves a trade-off between bandwidth occupancy (650~1100) and locality (the numbers of VM partitions of requests are all less than 2). Although the GBP algorithm guarantees the locality (the numbers of VM partitions are all less than 2), it is the worst in balancing the bandwidth occupancy (375~1500), which varies significantly. Let α be the ratio of maximum bandwidth occupancy over minimum bandwidth occupancy. As shown in Figure 2, α_{GBP} is 4, α_{LPTBP} is about 1.06, and α_{LLBP} is about 1.76. We originally set B_{L_c} infinity as a default. If we set B_{L_c} a value, the output placement schema might be changed. If $B_{L_c} \geqslant 825$ in LPTBP, the output VM placement schema would be not changed. When $B_{L_c} < 825$, more ToR switches are needed. If $B_{L_c} = 1000$ in LLBP, the new output VM placement schema is shown in Figure 2(d), in which the schema swaps a VM of R_3 for a VM of R_1. Though R_3 has more VM partitions, the bandwidth occupancy of uplinks is more balanced.

5. Evaluation

In this section, we evaluate the performance of the proposed heuristic algorithm (Least-Load First Based Placement, LLBP algorithm) by extensive simulations with different settings. We implement a Greedy Based Placement (GBP) algorithm and make it as the expected baseline of the algorithm performance. The Longest Processing Time Based Placement (LPTBP) algorithm can be taken as the optimal solution of the algorithm performance. We compare the performance of LLBP heuristic algorithm against the GBP and LPTBP algorithms. We first present the simulation settings and then show the evaluation results and corresponding analyses.

5.1. Simulation Settings. In our simulations, we set the five parameters as follows: the number of ToR switches nToR, the number of VMs that can be placed under each ToR switch nCap, the number of requests nReq, the number of VMs of each request nVMReq, and bandwidth requirements of each requested VMs nBnd. The scale of DCNs is denoted as nToR and nCap. In the simulations, we set $(1000, 80)$, $(1500, 60)$, $(2000, 40)$, and $(3000, 30)$. nVMReq is drawn from a uniform distribution between 80 and 120. nBnd is drawn from a normal distribution with a mean of μ and a variance of σ, where μ is drawn from a uniform distribution between 40 and 100 and σ is drawn from a uniform distribution between 0 and 40. nReq is determined by other parameters. We assume that all available VM slots are occupied by the VMs of requests.

5.2. Simulation Results. We run the GBP algorithm, LPTBP algorithm, and LLBP heuristic algorithm on the same randomly generated dataset under each scale of DCN for 100 rounds. We record the minimum bandwidth occupancy and maximum bandwidth occupancy on the uplinks of all the ToR switches for each round and get the average values shown in Figure 3. It is clear that the performance of the GBP algorithm is the worst, the LPTBP algorithm is the best,

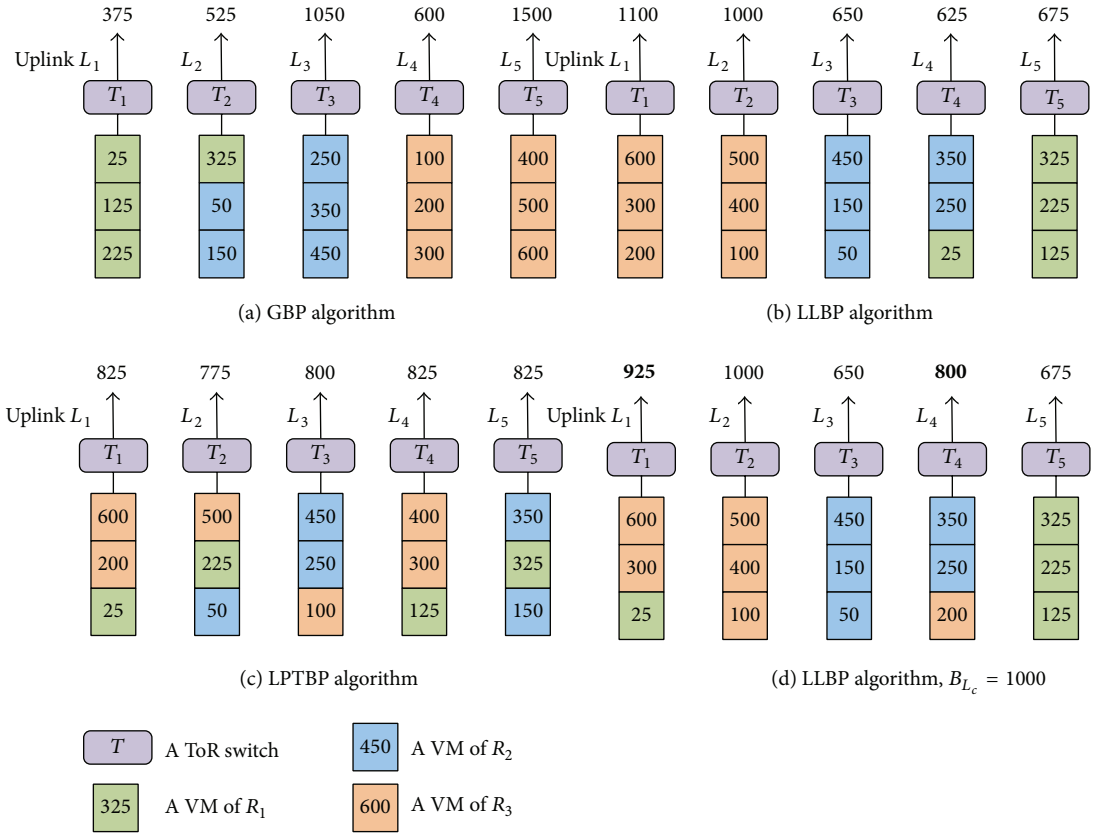


Figure 2: An example of comparison on the GBP, LLBP, and LPTBP algorithms.

and the LLBP algorithm is in the middle. According to the simulation results, α_{GBP} is larger than 3, α_{LPTBP} is nearly 1, and α_{LLFBP} is about 1.5. α values are close to the results of example in Section 4.

Although the GBP algorithm places as many as possible VMs of the same request under the same ToR switch, fully taking advantage of the communication locality property, the bandwidth occupancy on uplinks of all the ToR switches varies significantly. The LPTBP algorithm can spread the traffic of VMs evenly across all the uplinks of the ToR switches; however, it distributes the VMs of a request under several ToR switches. The proposed LLBP algorithm simultaneously balances the bandwidth occupancy on uplinks of all the ToR switches and takes advantage of the communication locality property. From the simulation results and analyses, we can conclude that LLBP algorithm performs effectively under different scales of DCNs.

6. Further Optimization

6.1. Locality. The classification of the topologies of data center networks (DCNs) falls under switch-centric, server-centric, and other topologies, according to their structural features. Switch-centric topologies consist of tree-like [6, 7, 18, 19], flat [20], and optical topologies [21, 22]. Server-centric topologies are divided into topologies designed for mega data centers [23, 24] and modular data centers (MDCs) [25, 26]. Other topologies include unstructured [27, 28] and wireless topologies [29, 30]. In DCNs, the locality can be defined as different levels. For example, in the fat-tree [6] in Figure 4, we can logically categorize locality into ToR level (S_0 and S_1), pod level (S_0 and S_2), and tree level (S_0 and S_4). Locality is different physically, such as server level, rack level, and row level. We can set different values for various locality levels in optimization.

6.2. Different Optimization Goal. In Section 3, we proposed Locality-Aware Virtual Machine Placement Problem with Traffic-Aware Balancing (LVMPPTB), which is a multiobjective optimization problem of simultaneously minimizing the maximum number of VM partitions of requests and minimizing the maximum bandwidth occupancy on uplinks of ToR switches. We also can optimize one objective while the other objective is fixed. For example, we can restrict the maximum bandwidth on uplink of a ToR switch to a fixed value to optimize the number of VM partitions of requests.

6.3. Online Balancing. VM placement problem occurs in the scenario of running applications from the very beginning, which is addressed by an offline algorithm. Once the VMs of applications are running, various problems could occur at different time, while we should use an online algorithm to perform VM migration for load balancing and fault tolerance.

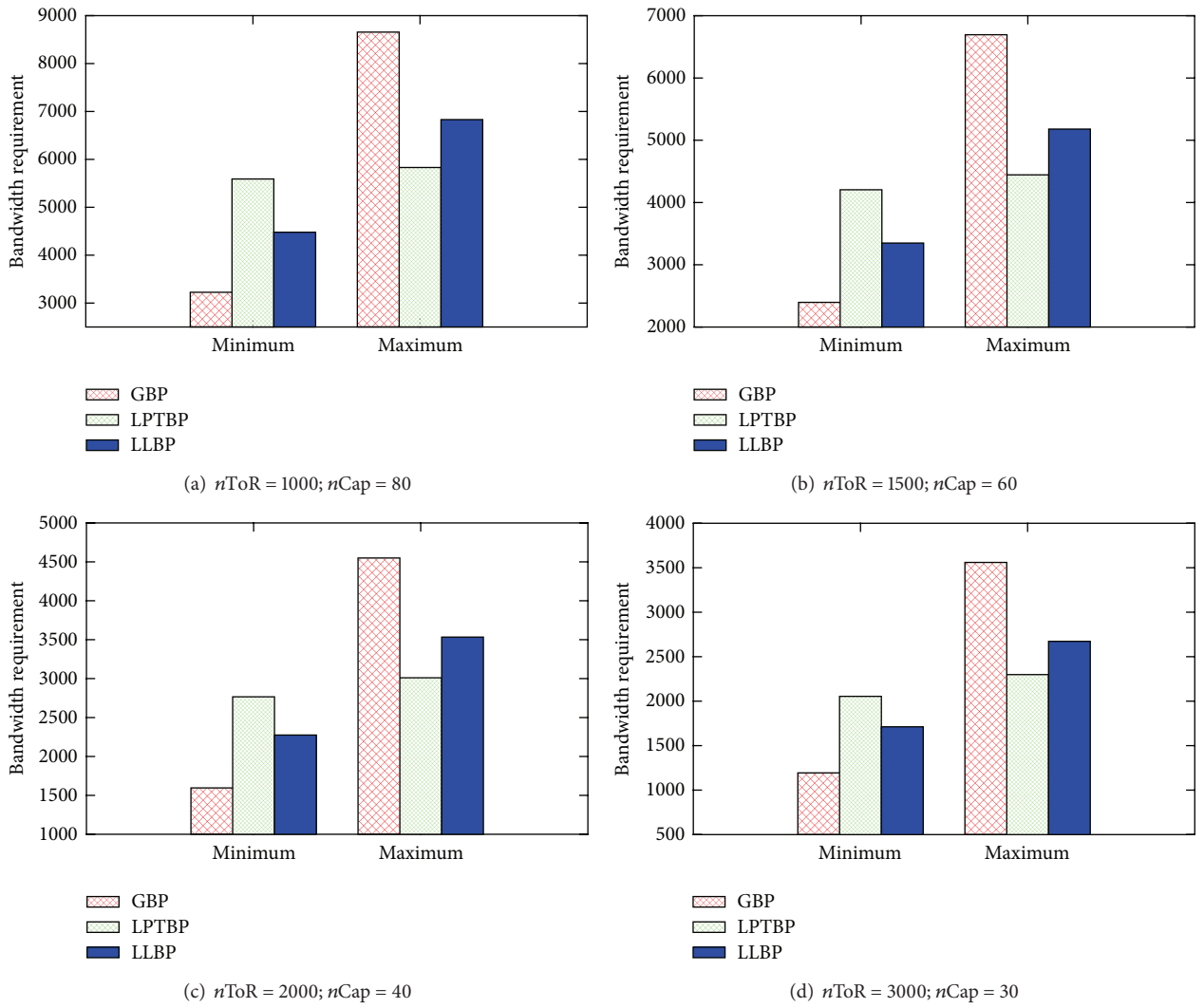


Figure 3: Minimum and maximum bandwidth requirement of different algorithms under different simulation settings.

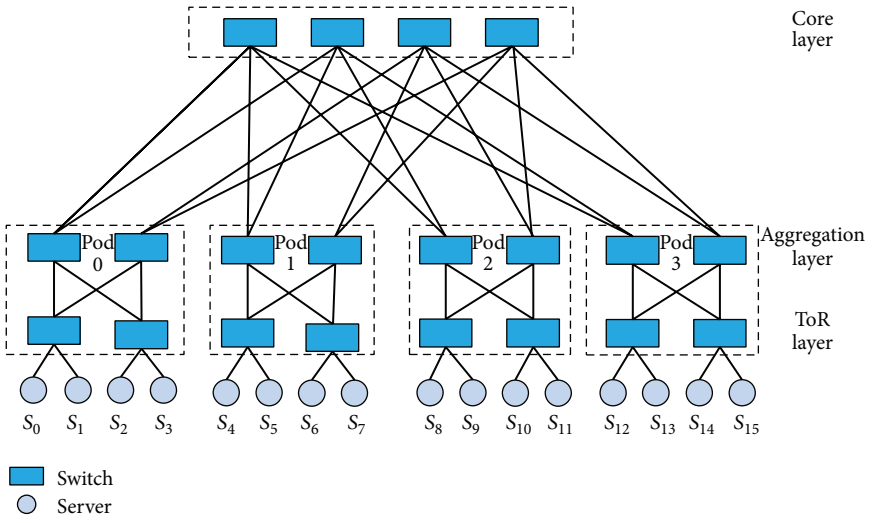


Figure 4: An example of fat-tree architecture.

We also can consider the correlations between several VMs to optimize the VM migration [31].

6.4. Fault Tolerance. In LPTBP and LLBP, we assumed that all requests of tenants were running normally. If the event of requests being lost occurs, we need to design a fault tolerant mechanism to retrieve the lost request. Timeout retransmission may be a way of solving the problem.

7. Conclusion

In this paper, we formulated Virtual Machine Placement Problem with Traffic Balancing (VMPPTB) to balance traffic in data centers. We proved its NP-hardness and designed a Longest Processing Time Based Placement algorithm. To take advantage of the communication locality property, we proposed Locality-Aware Virtual Machine Placement Problem with Traffic Balancing (LVMPPTB) which simultaneously minimizes the maximum number of VM partitions of each request and minimizes the maximum bandwidth occupancy on uplinks of ToRs. We proved its NP-hardness and designed a Least-Load First Based Placement heuristic algorithm. We conducted extensive simulations to evaluate the performance of algorithms.

Notations

$\mathbb{T} = \{T_k\}$: Set of ToR switches
c: Maximum number of VMs under a ToR switch
C_k: Number of VMs under ToR switch T_k
L_k: Uplink of ToR switch T_k
B_{L_k}: Accumulated bandwidth occupancy on uplink L_k
B_{L_c}: Maximum bandwidth capacity of uplink
$\mathbb{R} = \{R_i\}$: Set of requests from tenants
$\mathbb{S} = \{s_i\}$: Set of number of requested VMs of each request
B_{ij}: Bandwidth requirement of VM j of request R_i
D_{ij}^k: Indicator of whether VM j of request R_i is under T_k
$\mathbb{T}_{R_i}$: Set of ToR switches connected with VMs of request R_i.

Disclosure

The opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies or the government.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work has been supported in part by the China 973 Project (2014CB340303), China NSF Projects (nos. 61472252 and 61133006), the Opening Project of Key Lab of Information Network Security of Ministry of Public Security (the Third Research Institute of Ministry of Public Security) (Grant no. C15602), the Opening Project of Baidu (Grant no. 181515P005267), and the Open Project Program of Shanghai Key Laboratory of Data Science (no. 201609060001). The authors also would like to thank Tao Liang for his contributions on the early versions of this paper.

References

[1] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, vol. 37, no. 5, pp. 164–177, ACM, Lake George, NY, USA, October 2003.

[2] VMware Capacity Planer, http://www.vmware.com/products/capacity-planner.

[3] Novell PlateSpin Recon, http://www.novell.com/products/recon.

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] Cisco Data Center Infrastructure 2.5, http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND_2_5a_book.html.

[6] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[7] A. Greenberg, J. R. Hamilton, N. Jain et al., "Vl2: a scalable and flexible data center network," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 51–62, 2009.

[8] R. Niranjan Mysore, A. Pamboris, N. Farrington et al., "Portland: a scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 39–50, 2009.

[9] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of the IEEE INFOCOM*, pp. 1–9, IEEE, San Diego, Calif, USA, March 2010.

[10] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '12)*, pp. 2876–2880, Orlando, Fla, USA, March 2012.

[11] O. Biran, A. Corradi, M. Fanelli et al., "A stable network-aware VM placement for cloud systems," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '12)*, pp. 498–506, IEEE, Ottawa, Canada, May 2012.

[12] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference Internet Measurement (IMC '10)*, pp. 267–280, ACM, Melbourne, Australia, November 2010.

[13] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1990.

[15] K. You, B. Tang, and F. Ding, "Near-optimal virtual machine placement with product traffic pattern in data centers," in *Proceedings of the 2013 IEEE International Conference on Communications (ICC '13)*, pp. 3705–3709, IEEE, Budapest, Hungary, June 2013.

[16] X. Li, J. Wu, S. Tang, and S. Lu, "Let's stay together: towards traffic aware virtual machine placement in data centers," in *Proceedings of the 33rd IEEE Conference on Computer Communications (INFOCOM '14)*, pp. 1842–1850, Toronto, Canada, May 2014.

[17] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[18] B. Heller, S. Seetharaman, P. Mahadevan et al., "Elastictree: saving energy in data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI '10)*, p. 17, USENIX Association, 2010.

[19] M. Walraed-Sullivan, A. Vahdat, and K. Marzullo, "Aspen trees: balancing data center fault tolerance, scalability and cost," in *Proceedings of the 9th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT '13)*, pp. 85–96, December 2013.

[20] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 338–347, 2010.

[21] K. Chen, A. Singla, A. Singh et al., "OSA: an optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 498–511, 2014.

[22] K. Chen, X. Wen, X. Ma et al., "WaveCube: a scalable, fault-tolerant, high-performance optical data center architecture," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '15)*, pp. 1–9, Hong Kong, April-May 2015.

[23] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: a scalable and fault-tolerant network structure for data centers," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 75–86, 2008.

[24] D. Li and J. Wu, "On the design and analysis of data center network architectures for interconnecting dual-port servers," in *Proceedings of the IEEE Conference on Computer Communications (IEEE INFOCOM '14)*, pp. 1851–1859, Toronto, Canada, April-May 2014.

[25] C. Guo, G. Lu, D. Li et al., "BCube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.

[26] D. Li, M. Xu, H. Zhao, and X. Fu, "Building mega data center from heterogeneous containers," in *Proceedings of the 2011 19th IEEE International Conference on Network Protocols (ICNP '11)*, pp. 256–265, IEEE, Vancouver, Canada, October 2011.

[27] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: networking data centers randomly," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, pp. 1–14, San Jose, Calif, USA, April 2012.

[28] A. R. Curtis, T. Carpenter, M. Elsheikh, A. López-Ortiz, and S. Keshav, "REWIRE: an optimization-based framework for unstructured data center network design," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '12)*, pp. 1116–1124, Orlando, Fla, USA, March 2012.

[29] J.-Y. Shin, E. G. Sirer, H. Weatherspoon, and D. Kirovski, "On the feasibility of completely wireless datacenters," in *Proceedings of the 8th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '12)*, pp. 3–14, Austin, Tex, USA, October 2012.

[30] X. Zhou, Z. Zhang, Y. Zhu et al., "Mirror mirror on the ceiling: flexible wireless links for data centers," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 443–454, 2012.

[31] W. Wei, X. Wei, T. Chen, X. Gao, and G. Chen, "Dynamic correlative VM placement for quality-assured cloud service," in *Proceedings of the IEEE International Conference on Communications (ICC '13)*, pp. 2573–2577, IEEE, Budapest, Hungary, June 2013.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 3780163, 13 pages
http://dx.doi.org/10.1155/2016/3780163

Research Article

MultiCache: Multilayered Cache Implementation for I/O Virtualization

Jaechun No[1] and Sung-soon Park[2]

[1]*College of Electronics and Information Engineering, Sejong University, 98 Gunja-dong, Gwangjin-gu, Seoul 143-747, Republic of Korea*
[2]*Department of Computer Engineering, Anyang University and Gluesys Co. LTD, Anyang 5-dong, Manan-gu 430-714, Republic of Korea*

Correspondence should be addressed to Jaechun No; jano@sejong.edu

Received 16 February 2016; Revised 18 June 2016; Accepted 3 July 2016

Academic Editor: Zhihui Du

As the virtual machine technology is becoming the essential component in the cloud environment, VDI is receiving explosive attentions from IT market due to its advantages of easier software management, greater data protection, and lower expenses. However, I/O overhead is the critical obstacle to achieve high system performance in VDI. Reducing I/O overhead in the virtualization environment is not an easy task, because it requires scrutinizing multiple software layers of guest-to-hypervisor and also hypervisor-to-host. In this paper, we propose multilayered cache implementation, called MultiCache, which combines the guest-level I/O optimization with the hypervisor-level I/O optimization. The main objective of the guest-level optimization is to mitigate the I/O latency between the back end, shared storage, and the guest VM by utilizing history logs of I/O activities in VM. On the other hand, the hypervisor-level I/O optimization was implemented to minimize the latency caused by the "passing I/O path to the host" and the "contenting physical I/O device among VMs" on the same host server. We executed the performance measurement of MultiCache using the postmark benchmark to verify its effectiveness.

1. Introduction

Recently, VDI (Virtual Desktop Infrastructure) is becoming an essential aspect of the cloud-based computing environment due to its advantages such as user customization, easy-to-maintain software, and location-transparent accesses [1–3]. VDI multiplexes hardware resources of the host among VMs, which can improve server resource utilization and density. Also, VDI is capable of isolating VMs on the same host platform, which can offer the performance isolation and the secure application execution in the guest. This is performed by using the hypervisor that is responsible for coordinating VM operations and for managing physical resources of the host server.

While VDI offers several benefits, such as the increased resource utilization and the private data protection, there exist problems that can deteriorate the system performance, including I/O virtualization overhead [4, 5]. Before I/O requests issued in VMs are completed in VDI, they should go through multiple software layers, such as the layer from the back end, shared storage to the host server [6, 7], and the layers between guest operating system, hypervisor, and eventually host operating system.

Figure 1 shows the I/O virtualization path using KVM hypervisor and QEMU emulator. The application I/O requests are first handled by the guest kernel before being passed to the virtual, emulated device executing in the user space. After executing several modules including ones for the image format, those requests are entered to the host kernel by calling posix file system interface. The virtual disk is typically a regular file from the perspective view of the host file system. The files necessary for I/O requests can be stored either in the local disk attached to the host or in the shared storage connected by network.

As depicted in Figure 1, because the I/O virtualization path is organized with multiple software layers, optimizing I/O cost is very challenging, which requires scrutinizing various virtualization aspects. In this paper, we are interested in mitigating such an overhead by implementing the

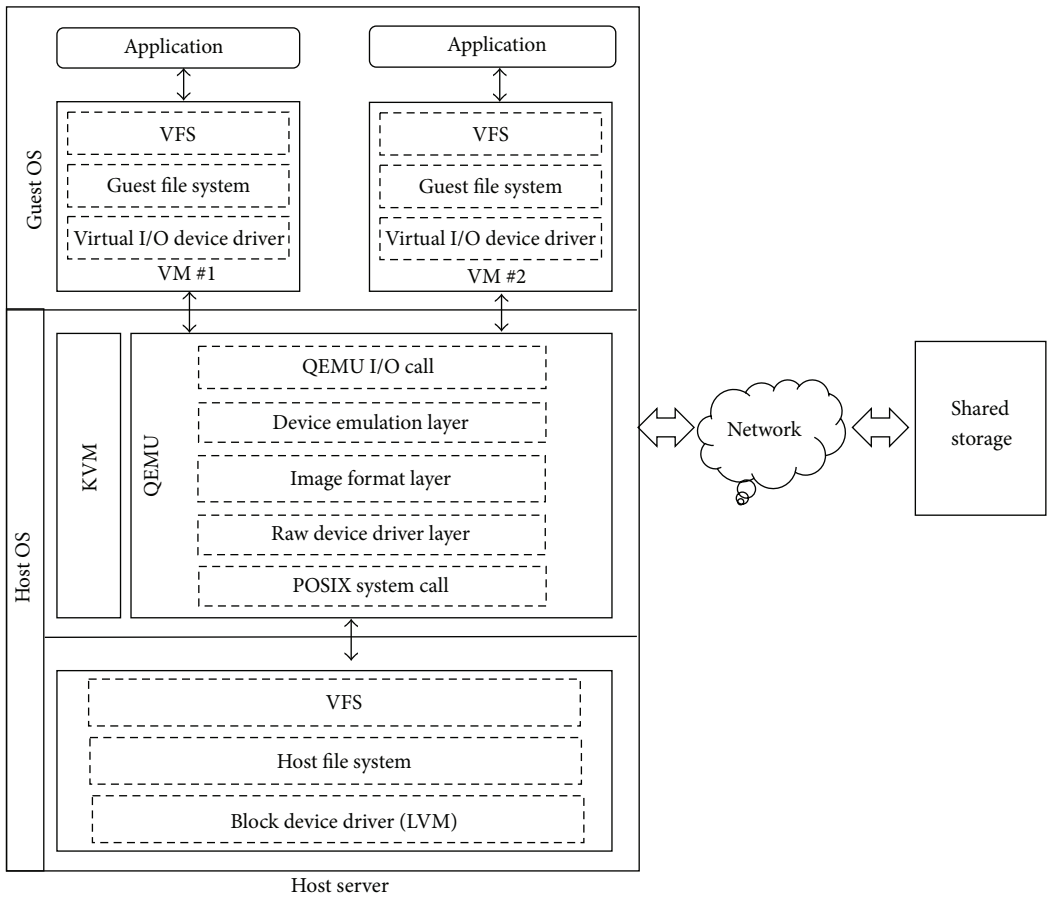


Figure 1: I/O virtualization path in KVM/QEMU.

appropriate cache mechanism in the guest and KVM using QEMU emulator [8–10].

Due to the thick software stack of VDI, implementing the virtualization cache method needs to take into account several layers with each I/O request passing through. For example, considering only the guest VM for the cache may not be enough to achieve the desirable I/O performance, because the latency occurring in the hypervisor, such as the context switching between the nonroot mode and the root mode, can substantially deteriorate application executions. Also, the OS dependency makes it difficult to port the guest-level cache method across VMs, especially in the case where VMs execute different guest operation systems.

In this paper, we propose the virtualization cache mechanism on top of KVM, called MultiCache (Multilevel virtualization Cache implementation), which combines VM's guest-level component with QEMU's hypervisor-level component. The main goal of the guest-level component of MultiCache is to alleviate the I/O overhead occurring in the file transmission between the back end, shared storage, and the guest. Also, caching on the guest level can give the better chance to retain the application-specific data. This is because while the guest needs to consider only the applications running on top of it, the hypervisor should control all the data necessary for VMs on the same host, which can cause the cache miss for the desired data due to the limited cache size or the swapping activity. Finally, by tightly coupling with the light-weight resource monitoring module, the component can manage the effective cache size in the guest.

The hypervisor-level component of MultiCache attempts to reduce I/O latency by supplying the desired data in QEMU instead of accessing the physical device of the host. The other contribution of the hypervisor-level component is to provide fast responsiveness by reducing the application process block time before I/O completion. The hypervisor mainly uses the hypercall to transit process control from the guest operation system to the hypervisor itself. Because such a transition requires the mode switching between nonroot mode and root mode, the application process on the guest should remain blocked, lagging the I/O performance behind. The hypervisor-level component tries to optimize such an overhead by providing the necessary data in QEMU.

This paper is organized as follows. In Section 2, we discuss the related studies and, in Section 3, we describe the overall structure of MultiCache. In Section 4, we present the performance measurement and, in Section 5, we conclude with a summary.

2. Related Studies

Reducing I/O virtualization cost is the critical issue to accelerate I/O bandwidth of virtual machines. There have

been several researches targeting I/O virtualization overhead. First of all, most VDI schemes use the back end and shared storage as a persistent data reservoir, such as DAS (host's direct attached storage), NAS (network-attached storage), or SAN (storage area network) [6, 11]. This storage is used to store read-only image templates or shared libraries and files for VMs. As the virtual machine has gained a widespread use in the cloud computing, managing the optimal cost for transferring the image contents and files between the storage and the host is becoming the essential research aspect. For example, Tang [6] proposed FVD (Fast Virtual Disk) consisting of VM image formats and the block device driver for QEMU. FVD enables supporting the instant VM creation and migration on the host by using copy-on-write, copy-on-read, and adaptive fetching.

As the technology of SSD (Solid State Disk) is rapidly growing, there have been several attempts to boost I/O bandwidth by adopting SSD in the virtualized environment [12–15]. For example, in vCacheShare [12], instead of proportionally allocating the flash cache space on the shared storage, vCacheShare uses the information about I/O accesses from VMs and trace processing data to extract reuse patterns in order to calculate the appropriate flash cache size. Mercury [13] is the client-server, write-through based flash cache method in the hypervisor. Byan et al. [13] argued that placing the flash cache either in the networked storage server or in VM may not be beneficial for speeding up I/O performance due to the network latency or VM migration [16, 17]. Also, utilizing flash cache with the write-back policy might not satisfy high I/O demand, because every write should still be written to the shared storage via a network hop for data consistency and availability.

S-CAVE [14] is a hypervisor-level flash cache to allocate the cache space among VMs. Similar to vCacheShare, S-CAVE monitors I/O activities of VMs at runtime and uses them to determine their cache space demand. Arteaga et al. [15] proposed a flash cache on the client-side storage system (VM host). They used dm-cache [18] block-level cache interface in their method and also argued that write-back policy is beneficial in the cloud environment. Razavi and Kielmann [19] tried to reduce the network overhead to be occurring during VM startup time, by placing VM cache either on the compute node or on the storage memory. They found that when the cloud environment supports a master image to be shared among multiple VMs, caching VM images on the compute node would efficiently reduce network traffics. Also, with the cloud environment, where many compute nodes simultaneously use multiple VMs, placing VM cache image to the storage memory can help reduce the disk queuing delay.

Besides between the shared storage and the host, the I/O latency taking place in the hypervisor should also be addressed to achieve the desirable system bandwidth in the virtualization environment. One of such overheads is VM exit. The I/O requests issued in VMs are asynchronously handled by the host while passing through the hypervisor and the emulator such as QEMU. Since VMs run on the nonroot mode and the hypervisor runs on the root mode, servicing I/O requests causes exiting VM first to go to the hypervisor, which incurs the context switching overhead. Also, the replies from the hypervisor to VMs adversely affect I/O performance. Since the application which issued those I/O requests remains blocked on VM, such a switching overhead can eventually slow down the application execution.

There are several researches on this issue. For example, SR-IOV [20, 21] was implemented to obtain the benefits of direct I/O on physical devices, by defining extensions to the PCIe specification. In SR-IOV, VDD running in the guest either is connected to VF executing on the shared sources for direct data movement or forwards the request to dom 0 where PF driver manages and coordinates the direct accesses to the shared resources for VFs. Yassour et al. [22] proposed a device assignment, where VM can access physical I/O resources directly, without passing through the host emulation software.

However, the direct device assignment cannot work for virtual resources such as virtual disk, losing the strength of virtualization flexibility. To overcome such a drawback, Har'El et al. [23] proposed a new form of paravirtual I/O, which tried to overcome the weakness of the existing paravirtual I/O scheme [4, 24, 25]. Their I/O scheme attempts to alleviate I/O overhead by providing the dedicated I/O core controlled by a single I/O thread. Instead of mixing I/O and guest workloads in the same core, using a dedicated I/O not only can assign more cycles to guests but also can improve overall system efficiency by reducing the context switching cost.

The other issue of the I/O virtualization overhead is that I/O requests should go through a thick I/O stack to complete. In the case of KVM using QEMU, the typical way of writing data in the guest is that, after passing the file system and device driver layer of the guest kernel, the data necessary for the write should be transferred to the emulated device driver in the hypervisor. Also, the data enters the host kernel that has the similar software structure to the guest kernel (assuming the guest and host run the same OS) and reaches the physical I/O device attached to the host. Appropriately placing cache is a way of reducing such traffics in the virtualization environment [26, 27].

Capo [27] uses local disks as a persistent cache. Shamma et al. insisted that the majority of requests on VMs are redundant and can be served by local disk. In order to justify their argument, they first traced a production VDI workload and found that caching below the individual VMs is effective to improve I/O performance. Capo was integrated with XenServer [28], by putting it into domain 0. Also. Gupta et al. [29] studied the page sharing and memory compression to save the memory consumption of VMs. Their difference engine method searches for the identical pages by using the hashing function. If pages have the same value, then it reclaims the pages and updates the virtual memory to point out the shared copy. Detecting the page sharing in their method goes further by eliminating the subpage sharing using page patching and by adapting in-core memory compression.

Ongaro et al. [30] studied the impact of Xen scheduling policy on I/O bandwidth with several applications showing the different performance characteristics. They found that

Xen's credit scheduler does not lower the response latency in the situation where several domains are concurrently performing I/O, even with BOOST state. One of the reasons is that the event channel driver always scans the pending vector from the beginning, instead of resuming from where it left. Also, they found the possibility of priority inversion of which delivering the highest-priority packet is postponed by preemption. Lu and Shen [31] traced the page miss ratio of VMs, by employing the hypervisor-level exclusive cache. They captured the pages evicted from VM memory into the hypervisor exclusive cache, while avoiding containing the same data in VM and exclusive cache. Jones et al. [32] also proposed a way of inferring promoting and evicting pages of buffer cache in the virtual memory. In order to correctly infer page cache activities, they observed some sensitive events causing control to be transferred to VMM, such as page faults, page table updates, and disk I/Os.

However, optimizing either in the guest or in the hypervisor might not be enough to produce the desirable performance because I/O path in the virtualization involves several software layers including the shared storage to guest and the guest to host. In this paper we attempted to target both layers by implementing the guest-level component and the hypervisor-level component.

3. MultiCache

3.1. System Structure. MultiCache was implemented to exploit I/O optimizations targeting multiple layers of I/O virtualization stack. Figure 2 represents an overall structure of MultiCache. As can be seen in the figure, MultiCache is divided into three components: guest-level component, hypervisor-level component, and resource monitoring component. The main goal of the guest-level component is to mitigate the I/O latency between the shared storage and the guest, by utilizing the history information of application I/O executions. Furthermore, by retaining the application-specific data in the guest, it can reduce I/O accesses to the physical device attached to the host. Finally, it tries to determine the effective cache size while taking into consideration VM and host resource usages in real time.

The guest-level component works at VM and consists of three tables, including hash table, history table and I/O map, to detect application's I/O activities and to retain the associated metadata representing the execution history logs. Those logs are used to predict the next I/O behaviour to preload the preferential files from the shared storage and also used to maintain recently referenced files in VM.

The hypervisor-level component was implemented in QEMU. The primary objective of this component is to minimize the I/O latency incurred in the virtual to hypervisor transition, by using the I/O access frequency measured in QEMU. Also, by intercepting I/O requests before they go to the host kernel, the component tries to reduce I/O contention among VMs. The first attribute of the component is the module interface interacting with QEMU I/O call while exchanging the associated I/O metadata with it, such as sector numbers requested. The main module of the component receives the I/O metadata from the interface and determines the hit or miss, while communicating with the metadata repository that contains the history logs of hypervisor's I/O execution, such as I/O access frequency. The device driver of the component is responsible for managing the hypervisor cache memory.

The third component of MultiCache is the real-time resource monitoring component. The monitoring module works at the hypervisor independently of guest operating systems, collecting the resource usage information from all VMs and the host server. The monitoring information is used by both components of MultiCache to effectively perform I/O optimization schemes. There are two tables associated with the monitoring component: VM resource table for storing VM resource usages and host resource table for host resource usages.

3.2. Differences between Two Components of MultiCache. There are four differences between the guest-level component and the hypervisor-level component of MultiCache. First, the main goal of the guest-level component is to mitigate I/O overhead between the shared storage and the guest VM, by prefetching and retaining files that will likely be used in the near future. On the other hand, the hypervisor-level component is to minimize I/O overhead between the guest VM and the host, by cutting down I/O software stack inside QEMU.

Second, two components of MultiCache use the different I/O unit: files in the guest-level component and sectors in the hypervisor-level component. While the guest-level component uses files for I/O optimization, the hypervisor-level component uses sectors that have been divided from files in the guest kernel before arriving at QEMU I/O call.

Third, to mitigate I/O overhead, the guest-level component utilizes the usage count that indicates how many times files have been referenced after they were brought into the guest. By caching the files that have high usage counts, the component attempts to reduce the network and I/O overheads between the shared storage and the guest VM. Also, this information is used to reduce I/O accesses from the host. The hypervisor-level component utilizes the I/O access frequency that implies how often sectors have been accessed from the host. Instead of forwarding sectors having frequently been used to the host, the hypervisor-level component caches those sectors in memory to reduce application process block time and I/O contention on the host.

Finally, while the guest-level component reserves the cache memory in the guest VM, the hypervisor-level component reserves the cache memory in the hypervisor, which is managed independently of guest operation systems. Table 1 illustrates the brief description about the differences between two MultiCache components.

3.3. MultiCache Guest-Level Component. The guest-level component of MultiCache was implemented to optimize network and I/O overheads incurring in file transmissions between the shared storage and the guest VM. Furthermore, by monitoring and accumulating I/O history information,

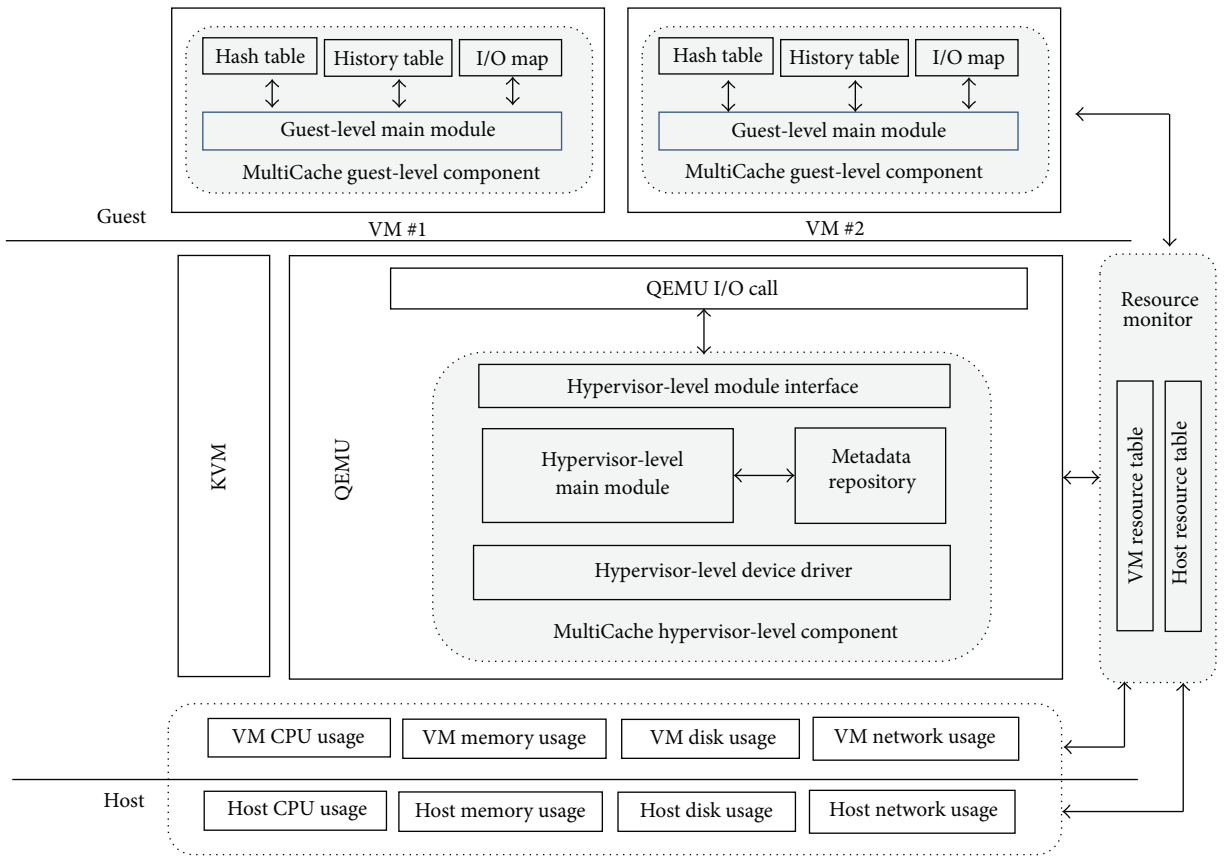


Figure 2: MultiCache structure.

Table 1: Difference between two MultiCache components.

	Guest-level component	Hypervisor-level component
Objective	To minimize the overhead between the shared storage and the guest VM	To minimize the overhead between the guest VM and the host
I/O unit	File	Sector
Optimization hint	File usage count	Sector I/O access frequency
Cache memory	Placed in the guest VM	Placed in the hypervisor
OS dependency	Yes	No

MultiCache enables providing better I/O responsiveness and data reliability.

To maintain the history information, MultiCache uses two kinds of tables: hash table and history table. The hash table is constructed with hash keys and is used to locate the associated history table containing the corresponding file metadata. There are λ history tables organized to solve the hash collision. One of the important file metadata in the history table is the usage count. Every time files are accessed for read and write operations, their associated usage count is increased by one to indicate the file access frequency. Also, MultiCache uses two I/O maps to determine the number of files to prefetch it from and to replace it to the shared storage.

Figure 3 shows the structure of MultiCache guest-level component. First, with the file inode, the hash key to access the hash table is calculated. The associated hash table entry contains the current history table address and its entry number where the desired file metadata can be retrieved. If the new file is used for I/O, then the next empty place in the current history table is provided to store its metadata.

In order to maintain the appropriate cache memory size in the guest, only the files with each having the usage count no less than *USAGE_THRESHOLD* are stored in the cache and their file metadata is inserted into the read or write map, based on file read or write operations. Separately maintaining read and write maps offers two benefits. First, it enables cashing more files showing frequent read executions in order to support the better chance for the fast read responsiveness. Second, it can contribute to enhancing data reliability and availability by flushing out more dirty files at the replacement phase. Besides, the I/O map enables maintaining files in the guest according to their frequency and recentness to reduce I/O accesses to the host.

In Figure 3, sections A (cache window size) and C in the read and write maps, respectively, illustrate the files that should be maintained in the cache memory; sections B and D are the candidates to be replaced under the cache memory pressure. MultiCache can enhance the read responsiveness

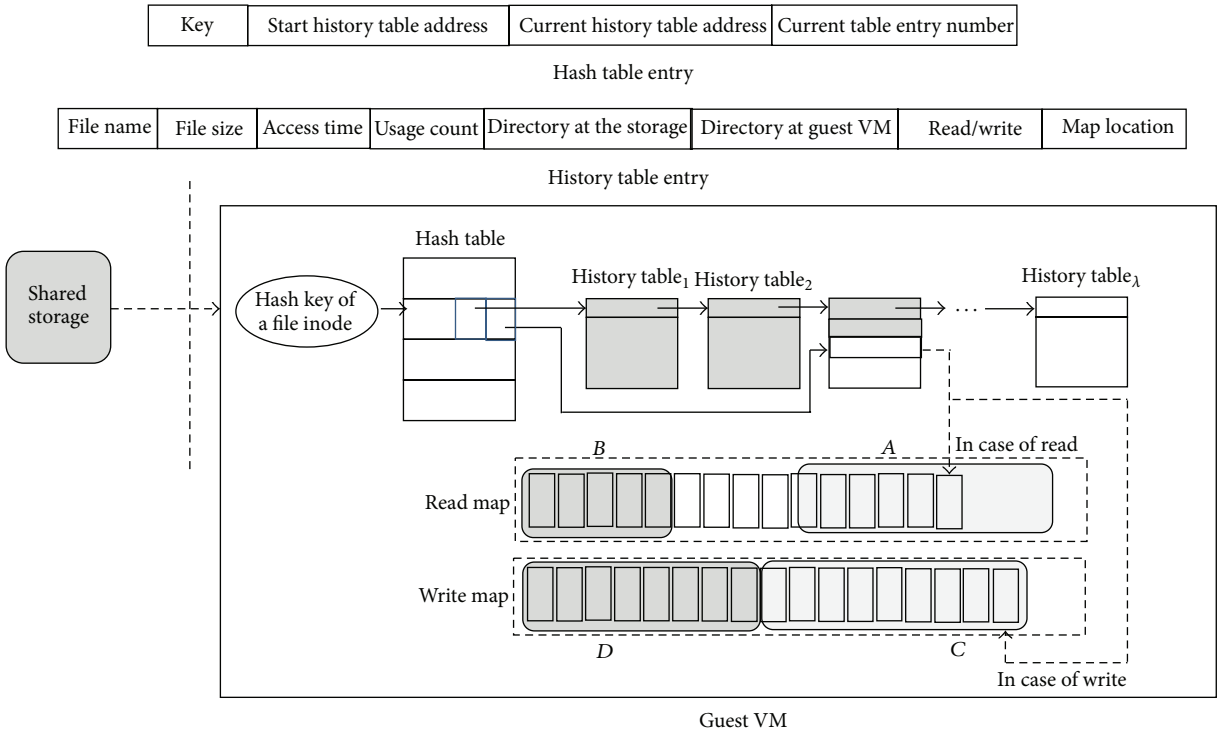


Figure 3: MultiCache guest-level component.

by caching more files whose most recent I/O accesses are read operations. Such a process involves replacing less files mapped in section B. Similarly, MultiCache can replace more dirty files mapped to section D for data reliability and availability. Let M_g be the guest-level cache memory size and let *MEM_THRESHOLD* be the memory usage limitation over which files designated at sections B and D must be flushed out to maintain the appropriate cache memory capacity. Finally, let f_a, f_b, g_c, and g_d be files whose metadata are mapped to sections A, B, C, and D, respectively. At each time epoch, MultiCache checks M_g by communicating with the resource monitor to see if the following condition is satisfied:

$$\frac{\left\{\sum_{f_a \in A} \text{size}\left(f_a\right) + \sum_{f_b \in B} \text{size}\left(f_b\right) + \sum_{g_c \in C} \text{size}\left(g_c\right) + \sum_{g_d \in D} \text{size}\left(g_d\right)\right\}}{M_g} \leq MEM\_THRESHOLD. \quad (1)$$

Algorithm 1 shows the steps involved in the guest-level component of MultiCache. Let i and k be the most recent positions of the read and write maps, respectively. Also, let f be the file for read and let g be the file for write.

In steps (1) and (2), MultiCache calculates the hash keys of f and g to access their file metadata from two tables. Also, the usage counts of two files are increased. In steps (4) to (17), if the usage count is larger than or equal to the threshold, then the metadata of f and g are inserted into the read and write maps, respectively, to store the associated data to MultiCache. In particular, if the last access of f was write, then the metadata is migrated from the write map to the read map, while erasing its history from the write map. The same procedure is applied for g to save its metadata to the write map. Steps (18) to (24) describe the procedure to maintain the appropriate cache size by taking into account condition (1). In the case that the condition is not satisfied, files mapped to sections B and D are flushed out to eliminate the memory pressure.

3.4. MultiCache Hypervisor-Level Component. The hypervisor-level component was implemented to minimize the I/O overhead caused by the software stack between the guest VM and the host. Before completing I/O requests, there are several mode transitions taking place between nonroot mode and root mode, which incurs the application execution being blocked. Furthermore, because those requests require accessing the data from the physical device attached to the host, the optimization at the hypervisor needs a way of reducing I/O contention on the device during the service time.

MultiCache hypervisor-level component uses several tables, called the metadata repository, to maintain I/O-related metadata at the hypervisor. Figure 4 shows the tables

(1) calculate the hash keys of f and g to retrieve their file metadata from the hash and history tables;
(2) **if** (not found) **then** insert their metadata to the hash table and the history table **end if**
(3) increase the usage counts of f and g by one;
(4) **if** (the usage count of $f \geq$ *USAGE_THRESHOLD*) **then**
(5) **if** ($f \in$ *read map* and its position in *read map* is a where $a < i$) **then**
(6) i++; move the metadata of f from ath position to ith position of *read map*;
(7) **else if** ($f \in$ *write map*) **then**
(8) i++; move the metadata of f to ith position of *read map*; delete it from *write map*;
(9) **else** i++; insert the metadata of f to ith position of *read map* **end if**
(10) **end if**
(11) **if** (the usage count of $g \geq$ *USAGE_THRESHOLD*) **then**
(12) **if** ($g \in$ *write map* and its position in *write map* is b where $b < k$) **then**
(13) k++; move the metadata of g from bth position to kth position of *write map*;
(14) **else if** ($g \in$ *read map*) **then**
(15) k++; move the metadata of g to kth position of *write map*; delete it from *read map*;
(16) **else** k++; insert the metadata of g to kth position of *write map* **end if**
(17) **end if**
(18) **for each time epoch**
(19) receive the current guest VM memory size from the resource monitor;
(20) check the condition specified in (1);
(21) **if** (the condition is not satisfied) **then**
(22) flush out files depicted in the section B or D until memory pressure is eliminated;
(23) **end if**
(24) **end for**

Algorithm 1: MultiCache guest-level component.

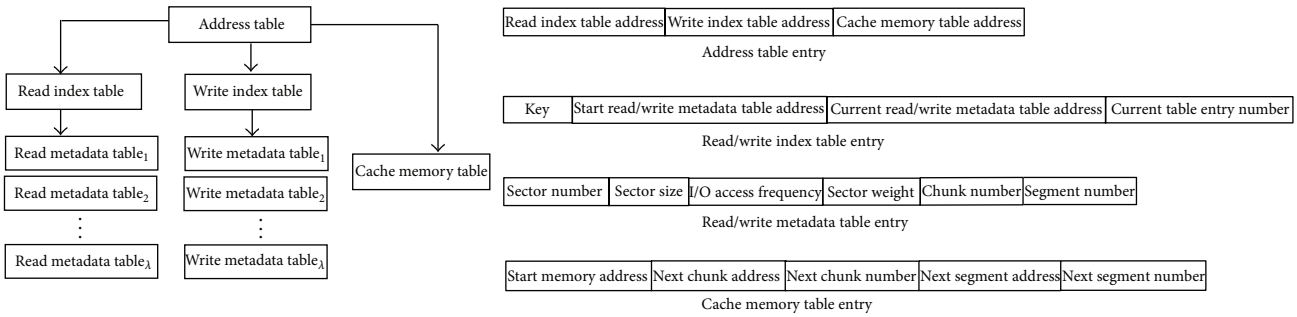


Figure 4: Metadata repository of the hypervisor-level component.

in the metadata repository. The address table stores the addresses of the read and write index tables containing the hash key and the start and current addresses of the read and write metadata tables. Similar to the history tables of the guest-level component, λ read metadata tables and λ write metadata tables are organized to target the collision problem. The read and write metadata tables contain the access information about the sectors transferred from QEMU I/O calls; the cache memory table maintains the next chunk and segment addresses of the hypervisor cache memory.

The hypervisor-level component uses I/O access frequencies of sectors to determine if those sectors should be retained in the cache memory. The I/O access frequency indicates how many times the associated sectors were used in I/O requests. There are two reasons for utilizing I/O access frequency. First, because the cache memory maintained in MultiCache is of a restricted size, a criterion is needed to filter sectors before storing them in the cache memory. In MultiCache, only those sectors that have been accessed no less than a threshold (*FREQ_THRESHOLD*) are stored in the cache memory.

Second, besides optimizing the mode transition and I/O contention aforementioned, MultiCache gives an opportunity to prioritize I/O requests, according to the VM's different importance. In other words, I/O requests issued in the high-priority guest VM can be executed first, despite their access frequency. In MultiCache, the priority of VM is determined by the number of CPUs and the memory capacity with which the VM was configured: the more number of CPUs and the larger memory size it is assigned, the higher priority the guest is given.

Let S be a set of sectors consisting of I/O requests in a guest. Consider a host where N number of VMs are currently executing. Also, each VM(i) is configured with u_i number of CPUs and v_i memory capacity.

Definition 1. A sector sc $\in S$ issued from VM(i) is defined by four components: p_{sc}, w_{sc}, δ_{sc}, and m_{sc}:

```
(1)  apply the hash function to obtain a hash key using sc;
(2)  access the read index table with the hash key to retrieve the metadata of sc from the read metadata table;
(3)  if no metadata about sc is available in the read index table then
(4)    store it in the read index table and the current read metadata table;
(5)    update the read index table to point out the next entry of the current read metadata table;
(6)  end if
(7)  p_sc++; w_sc = p_sc × the weight of guest VM;
(8)  if (w_sc < FREQ_THRESHOLD) then δ_sc = 0; exit to access sc from the host end if
(9)  if sc has not been mapped to the cache memory then
(10)   δ_sc = 0;
(11)   map sc to the cache, by retrieving the chunk and segment numbers from the cache memory table;
(12) else
(13)   δ_sc = 1;
(14)   access the cache memory with the chunk and segment numbers of sc retrieved;
(15) end if
```

Algorithm 2: MultiCache hypervisor-level component.

(1) p_{sc} is the I/O access frequency of sc.

(2) w_{sc} is the weight of sc satisfying $w_{sc} = p_{sc} \times (u_i / \sum_{k=1}^{N} u_k) \times (v_i / \sum_{k=1}^{N} v_k)$, where $(u_i / \sum_{k=1}^{N} u_k) \times (v_i / \sum_{k=1}^{N} v_k)$ is the weight of VM(i).

(3) δ_{sc} is the mapping function, indicating either cache hit ($\delta_{sc} = 1$) or miss ($\delta_{sc} = 0$).

(4) m_{sc} is the position of the cache memory, where sc is stored if $w_{sc} \geq FREQ\_THRESHOLD$.

Algorithm 2 represents the steps for reading sc at the hypervisor-level component of MultiCache.

Suppose that sc is one of the sectors consisting of a read request in the guest. MultiCache calculates a hash key to access the read index table containing the corresponding read metadata table address. After retrieving the associated metadata from the table, the I/O access frequency is multiplied by the VM weight to obtain the weight of sc. In the case where the weight of sc is less than *FREQ_THRESHOLD*, MultiCache passes sc to the host kernel to access it from the physical I/O device. Otherwise, from step (9) to step (15), MultiCache checks to see if sc has been stored in the cache memory. If not, sc is stored in the memory by using the chunk and segment numbers retrieved from the cache memory table. In the case where sc is found in the cache memory, it returns to the guest without going down to the host kernel. In the write operation, after updating the associated metadata to the write index table and the write metadata table, the sector is mapped to the cache table. If the associated metadata is available in the table, then the sector having been mapped in the cache memory is overwritten to update.

The cache memory handled by the hypervisor-level component is partitioned into chunks that consisted of a number of pages. In case of the write cache memory, the sectors stored in the chunk are transmitted to the host kernel, either after the chunk is filled with valid sectors or when the current checkpoint (currently every 30 seconds) for the chunk comes. Let M_h be the size of the cache memory; let C_i be the ith chunk; and let $|C_i|$ be the size of C_i. Also, let seg_k be the kth segment of C_i whose size, $|\text{seg}_k|$, is the same as that of a sector. The chunk validity and segment validity are determined by the chunk map and the segment map, respectively.

Definition 2. The allocation status of C_i in the chunk map and the one of seg_k of C_i in the segment map are defined as follows:

For any chunk C_i, bit: $C[i] \rightarrow \{0, 1\}$, $1 \leq i \leq M_h/|C_i|$.

For any segment $\text{seg}_k \in C_i$, bit: $\text{seg}[i, k] \rightarrow \{0, 1\}$, $1 \leq k \leq |C_i|/|\text{seg}_k|$.

If $\text{bit}(\text{seg}[i, k]) = 1$, then the segment contains a valid sector that should be transferred to the host. Otherwise, $\text{bit}(\text{seg}[i, k]) = 0$. Also, $\text{bit}(C[i]) = 1$ implies that all the segments consisting of C_i contain the valid sectors. Algorithm 3 shows the steps involved in the write process for the cache memory.

3.5. MultiCache Resource Monitor. The resource monitor calculates the resource statuses of guest VMs and host server at the hypervisor level because it should monitor the usage information independently of guest operating systems. Also, it is organized with the light-weight modules so that it rarely affects I/O bandwidth on VMs. During application executions, the monitor periodically notifies the resource usage information to the guest-level and hypervisor-level components to help them maintain the effective cache capacity for I/O improvement.

The resource monitor is composed of three modules: resource collection module, resource calculation module, and usage container. The resource collection module works on top of the server, while communicating with *proc* file system and *libvirt* to collect the resource status information such as CPU, memory, disk I/O, and network status. The resource calculation module calculates resource usages and, finally, the usage container stores the calculated information to offer it to both components of MultiCache.

Figure 5 represents the functions to be called in the resource monitor. To activate the monitor, the resource

(1) **for** each chunk C_i
(2) **if** (bit($C[i]$) = 1) **then** transfer C_i to the host; bit($C[i]$) = 0 **end if**
(3) **end for**
(4) /∗ let sc be the current sector to be stored in the cache memory ∗/
(5) /∗ let i and k be the chunk and segment numbers, respectively, selected from the cache memory table ∗/
(6) store sc to seg_k of C_i; bit(seg[i, k]) = 1; update bit($C[i]$) while reflecting bit(seg[i, k]) value;
(7) k++;
(8) **if** (($k < |C_i|/|\text{seg}_k|$) and bit(seg[i, k]) = 0) **then**
(9) store i and k to the cache memory table as the next position in the cache memory; exit;
(10) **end if**
(11) /∗ find the next empty chunk in the cache memory ∗/
(12) search for C_i such that ($i = (i + 1)\%(M_h/|C_i|)$ and bit($C[i]$) = 0);
(13) store i and 0 to the cache memory table as the next position in the cache memory;

Algorithm 3: Cache memory management.

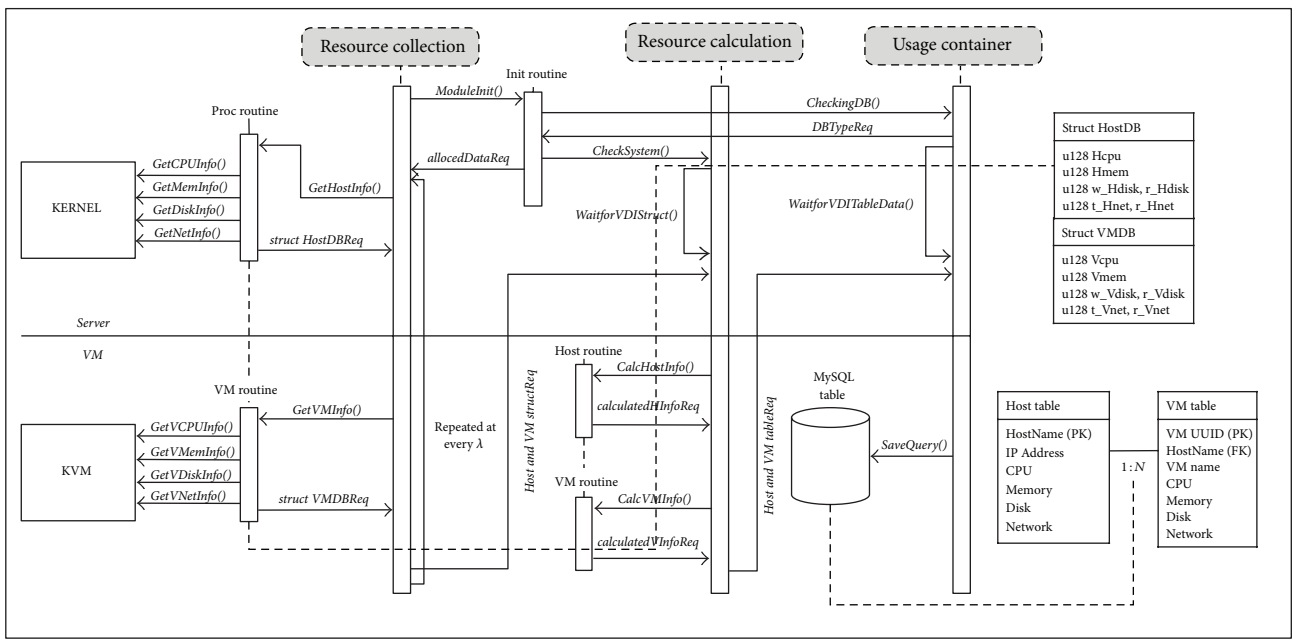


Figure 5: Resource monitor structure in MultiCache.

collection first initializes the functions to be called in the resource calculation and the usage container, by issuing *ModuleInit*( ). Also, it communicates with */proc* and *libvirt* at every time period λ by calling *GetXXXInfo*( ) and accumulates the resource status information to store it to the usage container. The resource calculation module retrieves the status information, by calling *calcHostInfo* and *calcVMInfo*( ), and calculates the resource usages by applying the formulas described in Table 2. Finally, the results are stored in the usage container.

4. Performance Evaluation

4.1. Experimental Platform. We executed all experiments on a host server equipped with an AMD FX 8350 eight-core processors, 24 GB of memory, and 1 TB of Seagate Barracuda ST1000DM003 disk. Also, the other server having the same hardware specification as the host server is configured as the shared storage node. Two servers are connected with 1 Gbit of network. The operating system was Ubuntu release 14.04 with 3.13.0-24 generic kernel. We installed the virtual machine on top of the host server by using KVM hypervisor. Each VM was configured with two-core processors, 8 GB of memory, and 50 GB of virtual disk using virtIO. The operating system of each VM was CentOS release 6.5 with 2.6.32-431 kernel. We used postmark benchmark for the evaluation.

4.2. MultiCache Guest-Level Component Evaluation. We first evaluated the guest-level component of MultiCache. In order to analyze the accurate I/O performance pattern of the guest-level component, we used the original KVM/QEMU version that is not integrated with the MultiCache hypervisor-level component. Also, we modified postmark to connect between the host server and the shared storage node. As a result, when files are generated from postmark, the files already brought into the guest from the storage node are read from

Table 2: Formulas and data structures for calculating resource usages.

Resource usage formula	Resource monitor data structure

$\text{CPU\%} = 100 - \frac{100}{\text{total}} \times (\text{idle}_{\text{now}} - \text{idle}_t)$
(i) Total: total CPU usage
(ii) idle_{now}, idle_t: idle values at the moment and at t
	struct S_ProcCpuInfo {
	_u128 user, nice, system, idle, iowait;
	_u128 irq, softirq, steal, guest;
	}

$\text{Memory usage} = 100 \times \frac{\text{total} - \text{free}}{\text{total}}$
(i) Total, free: total and free memory sizes, respectively
	struct S_ProcMemInfo {
	_u128 total, free;
	_u128 buffers, cached;
	}

$\text{Disk usage} = (\text{sectr}_{\text{now}} - \text{sectr}_t) \times 512$
(i) $\text{sectr}_{\text{now}}$, sectr_t: sector usages up to now and at t, respectively
	struct S_ProcDiskInfo {
	char disk_name[20];
	_u128 r_compl, r_merge, r_sectr, r_milsc;
	_u128 w_compl, w_merge;
	_u128 w_sectr, w_milsc;
	_u128 io_c_prc, io_milsc, io_w_milsc;
	}

Network usage
$\text{PPS} = \text{packet}_{\text{now}} - \text{packet}_t$
$\text{PacketSize} = \frac{(\text{byte}_{\text{now}} - \text{byte}_t) \times 8}{\text{PPS}} + 12 + 7 + 1$
$\text{BPS} = \text{PPS} \times \text{PacketSize}$
(i) PPS: number of packets transmitted per second
(ii) BPS: network bandwidth in bit per second
	struct S_ProcNetInfo {
	char net_name[20];
	char net_hwaddr[20];
	_u128 r_byte, r_pack, r_err, r_drop;
	_u128 r_fifo, r_frm, r_cmp, r_mult;
	_u128 t_byte, t_pack, t_err, t_drop;
	_u128 t_fifo, t_col, t_cal, t_cmp;
	}

MultiCache (*cached*) and the other files not residing in MultiCache are read from the storage through NFS (*not cached*).

Figure 6 shows I/O bandwidth while varying file sizes from 4 KB to 1 MB. x-axis represents the ratio of *not cached* to *cached*. For example, 90 : 10 implies that 90% of files to be needed during transactions are exchanged with the shared storage node. The number of transactions is 20000 and the ratio of read to write is 50 : 50. In the figure, as the percentage of files being accessed from MultiCache becomes high, better I/O bandwidth is achieved. Moreover, the effect of MultiCache is more apparent with large files. For example, with 20 : 80, where 80% of files are accessed from MultiCache, about 53% of I/O bandwidth improvement is observed with 1 MB of files as compared to that of 4 KB of files. The reason is that as more number of large files is accessed from MultiCache the network overhead to transfer data to VM becomes small, resulting in the bandwidth speedup.

In the evaluations, we observed that the effect of MultiCache is especially obvious with read operations, as shown in Figure 7. In order to see the impact of MultiCache in the mixed I/O operations, we varied the read and write percentages while increasing the number of transactions. In Figure 7, 80 : 20 means that 80% of transactions are read operations and 20% of transactions are writes. Also, we used 1 MB of file size. Figure 7 exhibits that better I/O throughput is generated with the large number of transactions and especially with the larger percentage of read operations. This is because write operations inevitably incur network and I/O overheads to store data to the shared storage and such burdens may lower the throughput.

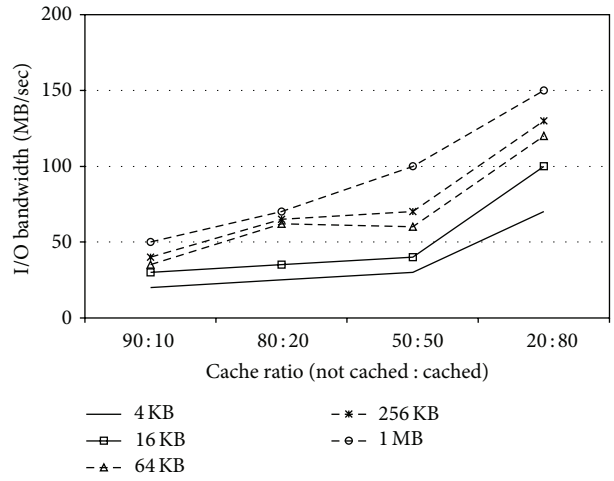


Figure 6: I/O bandwidth of virtCache.

However, I/O latency between the shared storage and the VM is not the only one that should be addressed to achieve the desirable performance. As mentioned, the I/O path from the guest to the host should also be scrutinized because there are multiple places causing the performance slowdown, such as I/O contention to physical devices and mode transition between the guest and the hypervisor. We will observe how the hypervisor-level component of MultiCache can achieve better bandwidth by overcoming such latencies.

4.3. I/O Bandwidth of Hypervisor-Level Component. We measured the I/O performance of the hypervisor-level component. In this experiment, there is no file transmission between

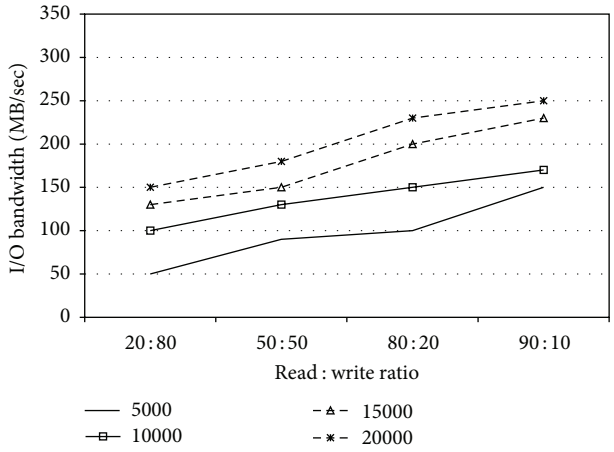


Figure 7: Performance evaluation based on I/O accesses.

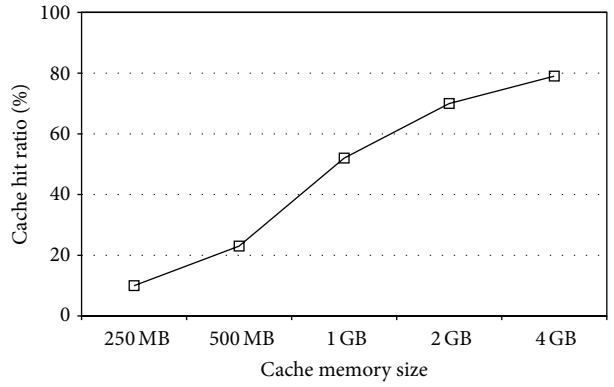


Figure 8: The effect of the cache memory.

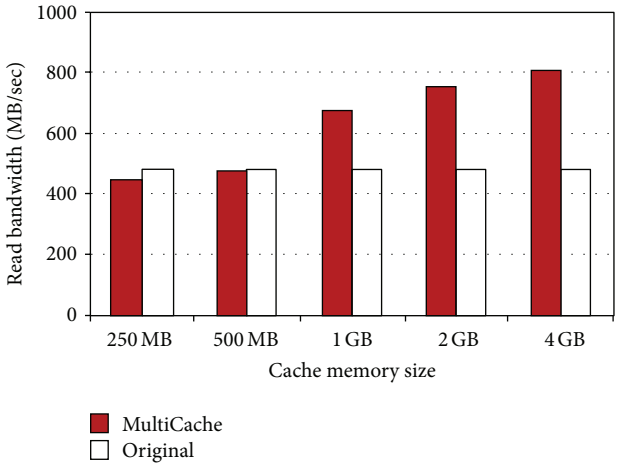


Figure 9: Read bandwidth based on the cache memory.

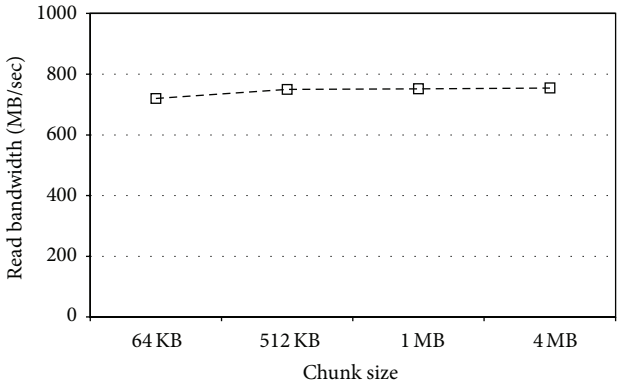


Figure 10: The effect of the chunk size on read.

the shared storage and the guest. In other words, all files for I/O were generated from the postmark benchmark running on the guest. The file sizes vary between 4 KB and 1 MB.

First of all, we observed the effect of the hypervisor cache memory in Figure 8, while changing the cache memory size from 250 MB to 4 GB. To warm the cache, we executed the modified postmark for 5 seconds and took the average value of each test case. Figure 8 shows the cache hit ratio obtained while changing the cache memory size. The figure shows that as the cache memory size becomes large, so does the hit ratio. For example, increasing the memory size from 250 MB to 4 GB shows the hit ratio improvement by up to 6.9x.

However, there is a subtle difference worthwhile to observe in the figure. While the hit ratio improves 126% from 500 MB to 1 GB, the hit ratio from 1 GB to 2 GB increases 34%. Extending the cache memory from 2 GB to 4 GB produces even the smaller percentage of hit ratio improvement. We guess that this is because the locality is shifted as time goes on. Also, the metadata stored in the metadata repository are replaced to the new ones due to the space restriction.

Figure 9 shows the I/O bandwidth obtained while varying the cache memory size from 250 MB to 4 GB. We can notice that Figure 9 depicts the similar performance pattern to that of Figure 8: the larger cache memory size is, the better I/O bandwidth is. Also, while increasing the cache memory size from 500 MB to 1 GB shows about 43% of bandwidth speedup, the cache memory extension from 1 GB to 2 GB produces only 12% of performance improvement.

In Figure 9, we compared the I/O performance of MultiCache to that of the original KVM/QEMU. With the small cache memory size such as 250 MB, the I/O bandwidth of MultiCache is less than that of the original version because of the cache miss incurred by space restriction. However, the performance difference becomes large as the memory size of MultiCache increases. With 2 GB of cache memory size, MultiCache produces about 37% of I/O bandwidth improvement compared to that of the original version. We currently use 2 GB of cache memory size. The RAM size of the host is 24 GB; therefore we use only about 8% of the total size as the cache memory.

Figure 10 shows the read results while changing the chunk size from 64 KB to 4 MB in the cache memory. As can be seen in the figure, changing the chunk size does little affect I/O bandwidth in the read operation because no write occurred to the host.

Figure 11 shows the write bandwidths of MultiCache while comparing to those of the original KVM/QEMU. The original version supports three write modes: default, write-through,

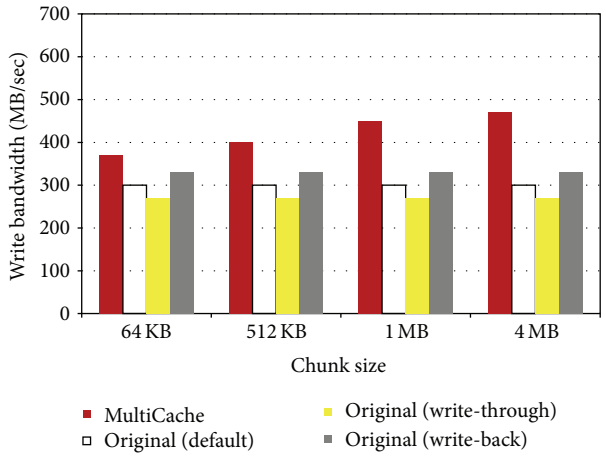


Figure 11: Write bandwidth based on the chunk size.

and write-back. In the case of using 1 MB of chunk size in MultiCache, it generates about 33% and 27% higher bandwidth than the default mode and the write-back mode of the original version, respectively. There are two reasons to explain such I/O bandwidth improvements. In the case of write-back mode of the original version, it buffers the data for I/O in the host kernel so that I/O requests issued in the guest should go through the guest kernel and QEMU before arriving at the host. Second, instead of flushing the data out to the physical device, the hypervisor-level component intercepts them in QEMU and collects in the cache memory in a big I/O unit. Such a method can contribute to accelerating I/O bandwidth, because, in Figure 11, we can notice that as the chunk size increases, the write performance also becomes large. However, based on the result with 4 MB of chunk size, increasing the size more than 1 MB might not produce the significant performance speedup due to the write latency in the host.

5. Conclusion

We proposed a multilayered cache mechanism, called MultiCache, to optimize I/O virtualization overhead. The first layer of MultiCache is the guest-level component whose main goal is to optimize the I/O overhead between the back end, shared storage, and the guest. Also, caching the application-specific data in the guest can contribute to accelerating the performance speedup. In order to achieve this goal, the guest-level component uses the history logs of file usage metadata to preload preferential files from the shared storage and to maintain recently referenced files in the guest. The second layer of MultiCache is to minimize the I/O latency between the guest and the host, by utilizing the I/O access frequency in QEMU. Also, by intercepting I/O requests in QEMU before they are transferred to the host kernel, the component can mitigate I/O contention on the physical device attached to the host. In the component, we accumulated the I/O access information about application executions in the metadata repository and used it to retain data with high I/O access frequency in the cache memory. Both components of MultiCache were integrated with the real-time resource monitoring module collecting the resource usage information of VMs and host at the hypervisor. The performance measurement with the postmark demonstrates that our approach is beneficial in achieving high I/O performance in the virtualization environment. As a future work, we will evaluate MultiCache with more real applications to prove its effectiveness in improving I/O performance.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2014R1A2A2A01002614).

References

[1] R. Spruijt, "VDI Smackdown," White Paper, v.1.4, 2012.

[2] J. Hwang and T. Wood, "Adaptive dynamic priority scheduling for virtual desktop infrastructures," in *Proceedings of the IEEE 20th International Workshop on Quality of Service (IWQoS '12)*, pp. 1–9, IEEE, Coimbra, Portugal, June 2012.

[3] D.-A. Dasilva, L. Liu, N. Bessis, and Y. Zhan, "Enabling green IT through building a virtual desktop infrastructure," in *Proceedings of the 8th International Conference on Semantics, Knowledge and Grids (SKG '12)*, pp. 32–38, Beijing, China, October 2012.

[4] J. Santos, Y. Turner, G. Janakiraman, and I. Pratt, "Bridging the Gap between Software and Hardware Techniques for I/O Virtualization," in *Proceedings of the USENIX Annual Technical Conference*, Boston, Mass, USA, 2008.

[5] Y. Dong, J. Dai, Z. Huang, H. Guan, K. Tian, and Y. Jiang, "Towards high-quality I/O virtualization," in *Proceedings of the Israeli Experimental Systems Conference (SYSTOR '09)*, article 12, May 2009.

[6] C. Tang, "FVD: a high-performance virtual machine image format for cloud," in *Proceedings of the USENIX Annual Technical Conference*, Portland, Ore, USA, June 2011.

[7] D. Le, H. Huang, and H. Wang, "Understanding performance implications of nested file systems in a virtualized environment," in *Proceedings of the 10th USENX Conference on File and Storage Technologies (FAST '12)*, San Jose, Calif, USA, February 2012.

[8] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: the Linux virtual machine monitor," in *Proceedings of the Ottawa Linux Symposium (OLS '07)*, pp. 225–230, July 2007.

[9] R. Russell, "Virtio: towards a De-Facto standard for virtual I/O devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[10] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, Anaheim, Calif, USA, April 2005.

[11] V. Tarasov, D. Hidebrand, G. Kuenning, and E. Zadok, "Virtual machine workloads: the case for new benchmarks for NAS," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST '13)*, pp. 307–320, Santa Clara, Calif, USA, 2013.

[12] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, "vCacheShare: automated server flash cache space management in a virtualization environment," in *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference (USENIX ATC '14)*, pp. 133–144, Philadelphia, Pa, USA, June 2014.

[13] S. Byan, J. Lentini, A. Madan et al., "Mercury: host-side flash caching for the data center," in *Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST '12)*, pp. 1–12, IEEE, San Diego, Calif, USA, April 2012.

[14] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu, and L. Zhou, "S-CAVE: effective SSD caching to improve virtual machine storage performance," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT '13)*, pp. 103–112, Edinburgh, UK, September 2013.

[15] D. Arteaga and M. Zhao, "Client-side flash caching for cloud systems," in *Proceedings of the 7th ACM International Systems and Storage Conference (SYSTOR '14)*, Haifa, Israel, June 2014.

[16] H. Jin, W. Gao, S. Wu, X. Shi, X. Wu, and F. Zhou, "Optimizing the live migration of virtual machine by CPU scheduling," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1088–1096, 2011.

[17] T. C. Ferreto, M. A. S. Netto, R. N. Calheiros, and C. A. F. De Rose, "Server consolidation with migration control for virtualized data centers," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1027–1034, 2011.

[18] dm-cache, http://visa.lab.asu.edu/dmcache.

[19] K. Razavi and T. Kielmann, "Scalable virtual machine deployment using VM image caches," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '13)*, Denver, Colo, USA, November 2013.

[20] S. Bhosale, A. Caldeira, B. Grabowski et al., *IBM Power Systems SR-IOV*, IBM Redpaper, IBM, 2014.

[21] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471–1480, 2012.

[22] B. Yassour, M. Ben-Yehuda, and O. Wasserman, "Direct device assignment for untrusted fully-virtualized virtual machines," Tech. Rep. H-0263, IBM Research, 2008.

[23] N. Har'El, A. Gordon, A. Landau, M. Ben-Yehuda, A. Traeger, and R. Ladelsky, "Efficient and scalable paravirtual I/O system," in *Proceedings of the USENIX Annual Technical Conference*, pp. 231–242, San Jose, Calif, USA, 2013.

[24] A. Menon, A. Cox, and W. Zwaenepoel, "Optimizing network virtualization in Xen," in *Proceedings of the USENIX Annual Technical Conference*, Boston, Mass, USA, 2006.

[25] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[26] H. Kim, H. Jo, and J. Lee, "XHive: efficient cooperative caching for virtual machines," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 106–119, 2011.

[27] M. Shamma, D. Meyer, J. Wires, M. Ivanova, N. Hutchinson, and A. Warfield, "Capo: recapitulating storage for virtual desktops," in *Proceedings of the 9th USENIX Conference on File and Storage Technologies*, San Jose, Calif, USA, February 2011.

[28] C.-H. Hong, Y.-P. Kim, S. Yoo, C.-Y. Lee, and C. Yoo, "Cache-aware virtual machine scheduling on multi-core architecture," *IEICE Transactions on Information and Systems*, vol. E95-D, no. 10, pp. 2377–2392, 2012.

[29] D. Gupta, S. Lee, M. Vrable et al., "Difference engine: harnessing memory redundancy in virtual machines," *Communications of the ACM*, vol. 53, no. 10, pp. 85–93, 2010.

[30] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors," in *Proceedings of the 4th International Conference on Virtual Execution Environments (VEE '08)*, pp. 1–10, Seattle, Wash, USA, March 2008.

[31] P. Lu and K. Shen, "Virtual machine memory access tracing with hypervisor exclusive cache," in *Proceedings of the USENIX Annual Technical Conference*, Santa Clara, Calif, USA, June 2007.

[32] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Geiger: monitoring the buffer cache in a virtual machine environment," *ACM SIGPLAN Notices*, vol. 40, no. 5, pp. 14–24, 2006.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 3976965, 14 pages
http://dx.doi.org/10.1155/2016/3976965

Research Article

VR-Cluster: Dynamic Migration for Resource Fragmentation Problem in Virtual Router Platform

Xianming Gao, Baosheng Wang, and Xiaozhe Zhang

College of Computer, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Xianming Gao; gxm9000@163.com

Received 27 December 2015; Revised 10 June 2016; Accepted 23 June 2016

Academic Editor: Ligang He

Network virtualization technology is regarded as one of gradual schemes to network architecture evolution. With the development of network functions virtualization, operators make lots of effort to achieve router virtualization by using general servers. In order to ensure high performance, virtual router platform usually adopts a cluster of general servers, which can be also regarded as a special cloud computing environment. However, due to frequent creation and deletion of router instances, it may generate lots of resource fragmentation to prevent platform from establishing new router instances. In order to solve "resource fragmentation problem," we firstly propose VR-Cluster, which introduces two extra function planes including switching plane and resource management plane. Switching plane is mainly used to support seamless migration of router instances without packet loss; resource management plane can dynamically move router instances from one server to another server by using VR-mapping algorithms. Besides, three VR-mapping algorithms including first-fit mapping algorithm, best-fit mapping algorithm, and worst-fit mapping algorithm are proposed based on VR-Cluster. At last, we establish VR-Cluster protosystem by using general X86 servers, evaluate its migration time, and further analyze advantages and disadvantages of our proposed VR-mapping algorithms to solve resource fragmentation problem.

1. Introduction

Network virtualization technology [1–3] has been regarded as a gradual solution to the development of Internet architecture, on which lots of researcher communities and equipment venders focus. In this context, service providers can rent network resources to lessees by establishing creating virtual networks. Meanwhile, they can recycle network resources occupied by virtual networks after the lifecycle of virtual networks comes to end. So, the future Internet can be regarded as "a big cloud computing environment," where operators can flexibly allocate and recycle network resources. For example, when one company wants to hold an online video conference, it can rent a virtual network for its business. In this way, users can get good quality of service during the lifecycle of virtual network. Besides, service provides can rent virtual network for research communities, and the latter can use these virtual networks to establish testbed and evaluate new network protocols in a real network environment.

Virtual router is one of key equipment to support the deployment of network virtualization technology, which is the same with traditional router in the Internet. Virtual router consists of several router instances, which can run in parallel and independently. And each router instance plays an important role in forwarding packets in virtual network. At the same time when service providers want to manage network infrastructure flexibly, network functions virtualization technology comes forth [4, 5]. The main idea behind it is to use general servers to establish network elements, such as virtual routers, firewall, NAT, and IDS. Besides, using general servers to establish packet processing devices catches more and more researchers' attentions, because the capability of packet processing in general server can be up to 40 Gbps by multicore and multithread technology. Under this environment, using a cluster of servers to achieve virtual router platform is a good solution [6]. So, service providers can flexibly create or release router instances in virtual router platform.

With service providers continually creating and deleting router instances, there is much discontiguous resource fragmentation in virtual router platform. Unfortunately, it cannot dispose of the incoming requests of creation of router instances any more, even though virtual router platform has enough resources. For instance, one platform consists of two servers with 10 Gbps processing ability: Server-A and Server-B. And service providers have established one router instance with 4 Gbps processing ability in Server-A and another router instance with 4 Gbps processing ability in Server-B. At the moment, it has two kinds of resource fragmentation on different servers: one resource block with 6 Gbps processing ability in Server-A and another resource block with 6 Gbps processing ability in Server-B. When a new request that is establishing router instance with 8 Gbps processing ability comes, it cannot create new router instance. Even though the remainder of resources in platform is up to 12 Gbps processing ability. In fact, resource fragmentation generated by frequent creation and deletion of router instances may prevent platform from responding to new requests when platform has enough resources. Thus, "resource fragmentation problem" is a severe problem. Unluckily, researchers do not pay any attention to the above problem, while putting lots of efforts into designing virtual router platform with high performance [7–11]. If we want to deploy virtual routers in the future, we should immediately start to attach importance to "resource fragmentation problem."

In this paper, we firstly put forward VR-Cluster platform to solve "resource fragmentation problem." It is established based on a cluster of servers by using the idea of network functions virtualization. From the aspect of function planes, it includes four function planes: switching plane, resource management plane, forwarding plane, and control plane. Forwarding plane and control plane also exist in the other virtual routers. Switching plane is a special plane, which uses flow strategies to transmit packets from exterior network to router instances and to transmit packets from router instances to exterior network. With the help of switching plane, operators can seamlessly migrate router instances from one server to another server without changing connection relationship between VR-Cluster platform and exterior network, which results in zero packet losses for migrated router instances. Resource management plane uses VR-mapping algorithm to dynamically calculate mapping relationship between router instances and physical platform and installs migration strategies into corresponding servers. And three VR-mapping algorithms including first-fit mapping algorithm, best-fit mapping algorithm, and worst-fit mapping algorithm are proposed, which are used to calculate mapping relationship between router instances and physical platform. At last, we establish a protosystem, namely, VR-Cluster, to support migration of router instances. VR-Cluster uses X86 servers to establish four function planes and uses switches to achieve full-mesh interconnection among function planes. Our experimental results show that VR-Cluster can migrate LF-Plane from one server to another server in one minute, which can meet requirements of dynamical migration. Besides, we also evaluate efficiency of our proposed VR-mapping algorithms based on resource model and evaluation model.

The remainder of this paper is organized as follow. Section 2 discusses and illustrates research work related to resource fragmentation in virtual router platform. Section 3 presents VR-Cluster architecture and exhibits how VR-Cluster supports dynamic migration. Section 4 establishes resource model for VR-Cluster and evaluation model for "resource fragmentation problem." Section 5 mainly exhibits three algorithms including first-fit mapping algorithm, best-fit mapping algorithm, and worst-fit mapping algorithm. Section 6 presents our experimental results including migration time and analysis of VR-mapping algorithms. Section 7 concludes this paper with a summary of our studies and discusses next works in the future.

2. Related Works

Many major router vendors and research communities have begun to follow suit in building support for router virtualization and explore how virtual routers can support multiple router instances running on the same underlying physical platform [2, 6–11]. It is a key device for bridging gap between network architectures and physical platforms. Thus, many people focus on moving toward virtual routers that can support polymorphic network architectures rather than monolithic network architectures and accommodate simultaneous coexistence of several router instances. However, unreasonable mapping relationship between router instances and virtual router platform may result in low resource utilization and increase the failure possibility of creation of router instances. We had explored it in our previous works [12].

Although "resource fragmentation problem" did not catch any researchers' attentions, some research communities had explored how router instances can move from one node to anther node [13–16]. For instances, RouterFarm [15] is proposed to support migration of router instances, which is achieved by reinstantiating a new router instance at a new location. However, operators need to reconfigure router instance, which may introduce downtime and packet loss. VROOM is another protosystem that is supporting for migration of router instances. In the earlier schemes of VROOM [16], which uses XEN to establish virtual environment and deploys router instance into each virtual environment, VROOM can move router instance from one location to another location by using XEN tools. In order to decrease downtime, current VROOM splits migration process into two steps: migration of control plane and migration of data plane. However, VROOM has to change network interfaces that is connecting to external network, which also results in changes of link status and recalculation of routing information.

However, existing schemes cannot achieve seamless migration of router instances, which may result in downtime and packet loss. In this paper, we try to use dynamical migration of router instances to solve "resource fragmentation problem." Thus, we should firstly put forward virtual router platform to support seamless migration before further analyzing VR-mapping algorithms.

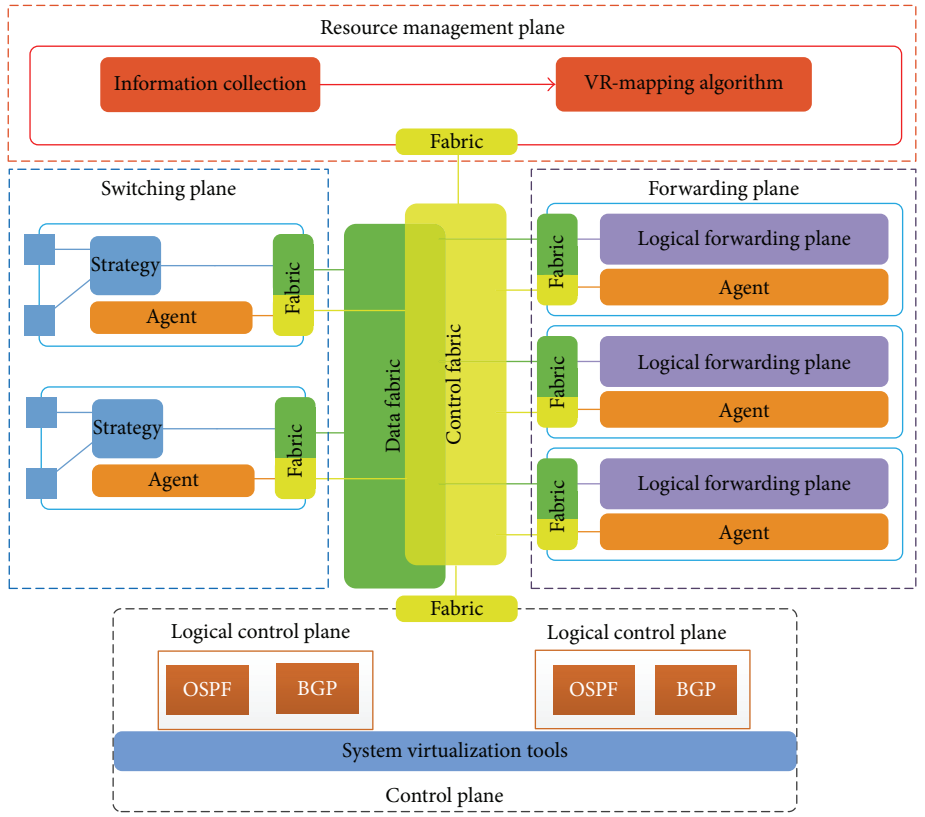


Figure 1: VR-Cluster platform architecture.

3. VR-Cluster

We mainly present VR-Cluster from two aspects: (1) platform architecture, which mainly includes four function planes, and (2) migration flow, which presents migration flow of router instance.

3.1. Platform Architecture. In order to support dynamic migration of router instances and to use dynamic migration to solve resource fragmentation problem, we put forward VR-Cluster architecture. Compared with other virtual router architectures, VR-Cluster includes four function planes: resource management plane, control plane, forwarding plane, and switching plane, as shown in Figure 1. Resource management plane and switching plane are customized function planes, which are used to support seamless migration of router instances.

VR-Cluster architecture mainly includes four layers: (1) resource management plane, which is responsible for managing platform, such as calculation of mapping relationship between router instances and physical platform, deployment of router instances, and release of occupied resources, (2) control plane, which usually uses system virtualization technology to run multiple logical control planes, (3) forwarding plane, which is used to run multiple logical forwarding planes, and (4) switching plane, which is responsible for connection between VR-Cluster and external network. VR-Cluster uses data fabric to achieve full-mesh interconnection between switching plane and forwarding plane. And control fabric is used to transmit control messages, such as configuration messages installed by resource management plane and control packets forwarded to control plane.

In VR-Cluster, forwarding plane and switching plane use "resource pool" idea, which consists of a cluster of servers. Forwarding plane can expand its forwarding ability by adding the quantity of servers; switching plane can flexibly change the amount and type of network interfaces. The high elasticity of VR-Cluster is brought by interconnection network including data fabric and control fabric. Thus, VR-Cluster not only can strongly increase management flexibility, but also has a good expansibility to support its ability.

3.1.1. Resource Management Plane. Resource management plane is responsible for management tasks in VR-Cluster. It has two main components: information collection (IC) and VR-mapping algorithm (VR-MA), as shown in Figure 2. And it usually runs in a single server, which can interconnect with other function planes via control fabric.

IC component is mainly responsible for collecting physical information and VR information from other three function planes. For physical information, it needs to know the quantity of servers in each function plane and the amount of each resource in servers. For VR information, it should know the amount of router instances, the occupied resources of router instances, and the location of router instances. IC component periodically collects VR-Cluster information and notifies VR-MA component with this information.

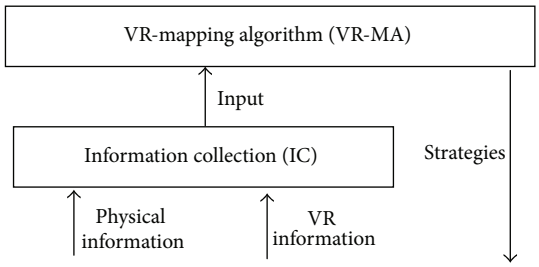


Figure 2: Resource management plane.

VR-MA component is used to calculate mapping relationship between router instances and physical platform based on collected information. After it completes calculation of migration schemes, it will send migration strategies to corresponding servers and execute migration actions. In this paper, we propose three VR-mapping algorithms and will present them later.

3.1.2. Control Plane. Control plane runs multiple logical control planes (LC-Planes for short) in parallel by using system virtualization technology, such as VMware [17], XEN [18], LXC [19], and KVM [20]. These LC-Planes share the same physical resources. And each LC-Plane can run either routing demos, such as Quagga [21] and XORP [22], or customized protocols to support new network architecture. Compared with forwarding plane and switching plane, control plane does not need to adopt "resource pool" idea, because a single server can support more than one hundred virtual environments in XEN or LXC. Although operators frequently create or recycle LC-Plane, it may not produce any resource fragmentation. Besides, the migration of logical control planes does not avoid downtime of router instances. Thus, VR-Cluster does not support migration of control plane. We think that migration of control plane just increases complexity of VR-Cluster, rather than resource utilization ratio.

Although VR-Cluster does not need to migrate control plane, it should cooperate with migration of forwarding plane. When one new logical forwarding plane is reinstantiated and sends registration messages to its corresponding LC-Plane, the latter will install routing tables and other configurations into this LF-Plane. And when the older LF-Plane is recycled by resource management plane, LC-Plane will not send any message to the older.

3.1.3. Forwarding Plane. Forwarding plane runs multiple logical forwarding planes (LF-Planes for short) in parallel. It does not use system virtualization technology to virtualize data plane, because using system virtualization to achieve forwarding plane has low forwarding ability. Now, each LF-Plane usually adopts function-based router (such as Click [23]) or flow-based router (such as OpenSwitch [24]). Each LF-Plane must have a corresponding LC-Plane in control plane.

Forwarding plane consists of a cluster of servers to meet various requests of forwarding ability in router instances. Resource fragmentation problem mainly results by creation and deletion of LF-Planes, rather than LC-Plane. At the same time, forwarding plane does not directly connect with exterior networks. When one LF-Plane is moved from one server to another server, it does not care about change of network interfaces. In other virtual router schemes, operators should adjust network interfaces that are connecting with exterior network while moving router instances, which does not avoid downtime of router instances. In VR-Cluster, forwarding plane and switching plane separation is a key design that can support seamless migration of forwarding plane.

When resource management plane installs migration strategies into forwarding plane, a new LF-Plane is established in forwarding plane. At the same time, LF-Plane should send registration messages to its corresponding LC-Plane and gets overall routing tables and other configurations from LC-Plane. Before new LF-Plane completes above action, the old LF-Plane is running in forwarding plane and is responsible for packet forwarding. Once resource management plane completes installation of routing tables, it configures switching plane, makes packets received from exterior network become sent to new LF-Plane, and recycles physical resources occupied by old LF-Plane.

3.1.4. Switching Plane. Switching plane is a special plane, which is used to provide network interfaces to connect with exterior network. Switching plane also adopts "resource pool" idea, as well as forwarding plane. Operators can expand the amount of network interfaces by adding servers in switching plane.

It adopts link virtualization technology to logically split physical network interface into several virtual network interfaces. Resource management plane establishes virtual network interfaces in switching plane and allocates these interfaces to corresponding router instances. Packets are continually received by virtual network interfaces in switching plane, and the latter sends packets to forwarding plane through data fabric based on flow strategies. Similarly, when forwarding plane forwards packets to exterior network, it firstly transmits packets to switching plane through data fabric and switching plane is responsible for packet transmission. Thus, operators can arbitrarily design their LF-Plane without caring about network interfaces that connected with exterior network.

Once these virtual interfaces are created, their lifecycle lasts until router instances are recycled by resource management plane, and resource management plane cannot move virtual network interface from one server to another server, as shown in Figure 3. If resource management plane changes network interface in one router instance, link status also changes, which can result in recalculation of routing tables. In some special scenes, it can breakdown end-to-end communication, which does not go against original idea in migration of router instances. Thus, VR-Cluster does not migrate network interface in router instances.

3.2. Migration Flow. Due to private features of virtual router, VR-Cluster just migrates LF-Plane, rather than the overall router instances, because it does not result in downtime of router instances and recalculation of routing tables. In this

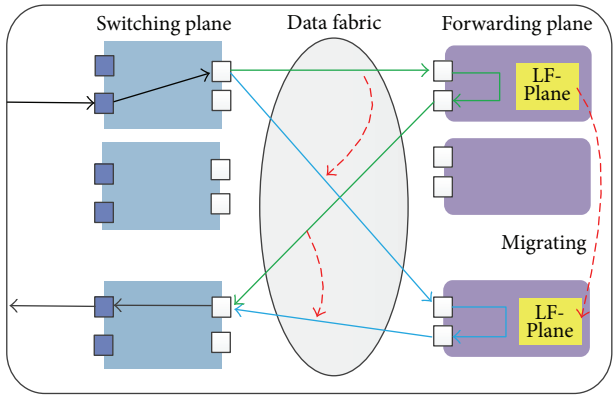


Figure 3: Interconnection relationship between switching plane and forwarding plane.

part, we mainly present how VR-Cluster can achieve seamless migration of router instances, as shown in Figure 4.

The procedure of migration of router instances in VR-Cluster mainly includes four steps, and tasks in each step are as shown in the following.

Step 1. There is a router instance in VR-Cluster. Packets received from exterior network are sent to LF-Plane through data fabric. If packets belong to control packets, they will be sent to LC-Plane by LF-Plane through control fabric. Besides, packets forwarded (or generated) by LF-Plane (or LC-Plane) must be sent to exterior network by switching plane. Resource management plane is responsible for checking whether or not there are migrating router instances, as shown in Figure 4(a).

Step 2. Resource management plane firstly establishes a new LF-Plane. Even though there are two LF-Planes, packets received by switching plane are just sent to old LF-Plane, rather than new LF-Plane. After resource management plane creates new LF-Plane, it immediately installs routing tables and configurations into new LF-Plane. Only when new LF-Plane has the same status as old LF-Plane, new LF-Plane has a chance to forward packets, as shown in Figure 4(b).

Step 3. Once when resource management plane has completed installation of routing tables and configurations, it will configure switching plane and control plane. And switching plane will send packets to new LF-Plane, instead of old LF-Plane. At the same time, it also configures control plane and makes the latter transmit control packets to new LF-Plane. However, resource management plane does not recycle old LF-Plane, because old LF-Plane also has some packets to not be disposed. Resource management plane does not recycle old LF-Plane until old LF-Plane has completed packet forwarding, as shown in Figure 4(c).

Step 4. Resource management plane recycles old LF-Plane. During migration flow of router instance, network interfaces do not change. LF-Plane may move from one server to another server. In this context, VR-Cluster completes seamless migration of router instance, as shown in Figure 4(d).

4. Mathematics Model

In this section, we firstly establish resource model for VR-Cluster and analyze main resources in each function plane. And then we establish evaluation model for resource fragmentation problem based on resource model in VR-Cluster.

4.1. Resource Model. We firstly analyze fundamental resource in VR-Cluster before we establish evaluation model for "resource fragmentation problem." VR-Cluster mainly includes three physical resources: control resource, forwarding resource, and switching plane. The three types of physical resources are responsible for different function planes. Besides, resource model should also include resources occupied by router instances. So resource model mainly includes four types of resource as follows.

Control Plane Resource. It refers to physical resources in control plane, which determines whether available resources meet requirements of LC-Plane. Cx refers to physical resources in control server that run LC-Planes. Each Cx includes three parts of resources: (1) CPU, which is responsible for calculation applications; (2) link-bandwidth, which is mainly used to communicate with its corresponding LF-Plane via control fabric; and (3) memory, which is used to store related information, such as link-state database, routing tables, and configuration rules, as shown in the following formula:

$$Cx = \sum \left(C_{\text{cpu}} + C_{\text{link}} + C_{\text{mem}}\right). \tag{1}$$

And C refers to the total physical resources in control plane, which is calculated as shown in the following formula:

$$C = \sum_{i=1}^{N} Ci \quad (i \in c). \tag{2}$$

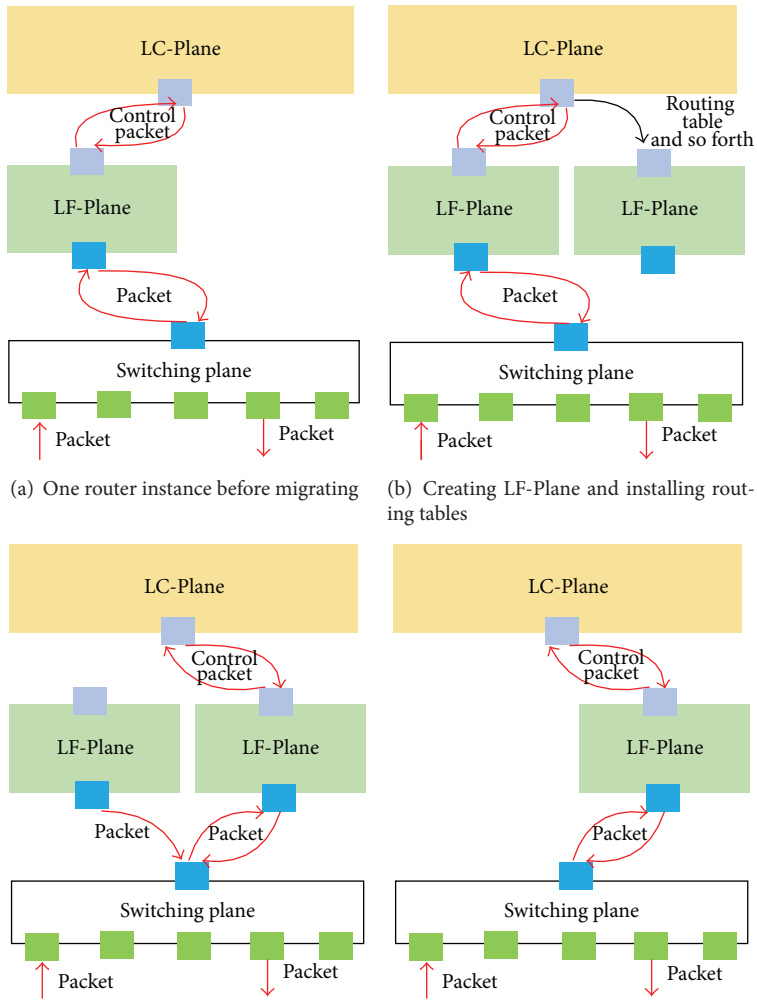
Forwarding Plane Resource. It refers to forwarding resources in forwarding plane. Fx refers to physical resource in forwarding server that runs LF-Planes. Each Fx also includes three parts of resources: (1) CPU, which is responsible for forwarding applications; (2) link-bandwidth, which is used to communicate with its corresponding LC-Plane via control fabric and to interconnect with switching plane via data fabric; and (3) memory, which is used to store some related information, such as forwarding tables or flow tables, as shown in the following formula:

$$Fx = \sum \left(F_{\text{cpu}} + F_{\text{link}} + F_{\text{mem}}\right). \tag{3}$$

And F refers to the total physical resources in forwarding plane, which is calculated as shown in the following formula:

$$F = \sum_{i=1}^{N} Fi \quad (i \in f). \tag{4}$$

Switching Plane Resource. It refers to network interfaces in switching plane. Sx refers to physical resource in switching



(a) One router instance before migrating

(b) Creating LF-Plane and installing routing tables

(c) Packets sent to new LF-Plane and reserving old LF-Plane

(d) Recycling old LF-Plane

Figure 4: Migration flow of VR-Cluster.

server that provides network interfaces interconnected with exterior network for router instances. Each Sx mainly consists of two types of network interfaces: (1) exterior interface, which is used to connect with exterior network, and (2) fabric interface, which interconnects with forwarding plane via data fabric, as shown in the following formula:

$$Sx = \sum \left(S_{\text{exterior}} + S_{\text{fabric}}\right). \tag{5}$$

And S refers to the total physical resources in switching plane, which is calculated as shown in the following formula:

$$S = \sum_{i=1}^{N} Si \quad (i \in f). \tag{6}$$

Router Instance Resource. It refers to physical resources occupied by router instances in VR-Cluster. Rx refers to physical resources occupied by one router instance, which includes three parts: (1) R_C^x, which stands for control resources occupied by LC-Plane in router instance x, (2) R_F^x, which stands for forwarding resources occupied by LF-Plane in router instance x, and (3) R_S^x, which refers to interfaces occupied by router instances x, as shown in the following formula:

$$Rx = R_C^x + R_F^x + R_S^x. \tag{7}$$

And R refers to the total physical resources occupied by router instances, which is calculated as shown in the following formula:

$$R = \sum_{i=1}^{N} Ri \quad (i \in VR). \tag{8}$$

We can use four types of resource to calculate resource utilization in VR-Cluster that is an important criterion. The resource utilization refers to the ratio of resources occupied by router instances to overall physical resources including overall control resources, overall forwarding resources, and

overall interface-resources, as shown in the following formula:

$$R_{\text{utilization}} = \frac{R}{C + F + S}. \tag{9}$$

4.2. Evaluation Model. In order to evaluate "resource fragmentation problem" in VR-Cluster, we put forward three evaluation criterions: (1) failure magnitude of creation of router instances, (2) magnitude of resource fragmentation, and (3) ratio of resource fragmentation. The details and calculation formulas of these three evaluation criterions are presented as follows.

4.2.1. Failure Magnitude of Creation of Router Instances. This evaluation criterion is used to record failure times in the processing of creation of router instances. Resource management plane cannot establish new router instances, when it meets one of three conditions as shown in the following formula:

$$\text{sum}(\text{fail}) = \begin{cases} \text{sum}(\text{fail}) + 1 & \text{if } \left(\left(R_C^x > \max\{\text{Re}\, m(Cx)\}\right) \&\& \left(R_C^x < C\right)\right) \\ & \text{if } \left(\left(R_F^x > \max\{\text{Re}\, m(Fx)\}\right) \&\& \left(R_C^x < F\right)\right) \\ & \text{if } \left(\left(R_S^x > \max\{\text{Re}\, m(Sx)\}\right) \&\& \left(R_C^x < S\right)\right) \\ \text{sum}(\text{fail}) & \text{other.} \end{cases} \tag{10}$$

In Formula (10), $\max\{\text{Re}\, m(Cx)\}$ refers to the maximum of the remainder of resource block in control plane; $\max\{\text{Re}\, m(Fx)\}$ refers to the maximum of remainder of resource block in forwarding plane; and $\max\{\text{Re}\, m(Sx)\}$ refers to the maximum of remainder of interface in switching plane.

Formula (10) is mainly used to record failure magnitude of creation of router instances, when VR-Cluster has enough physical resources exceeding requested resources. If this evaluation criterion is too big, VR-Cluster has a severe resource fragmentation problem. Otherwise, operators can establish router instances with a low possibility of failure of creation of router instances. Thus, we can use Formula (10) to reflect resource fragmentation problem in VR-Cluster.

4.2.2. Magnitude of Resource Fragmentation. This evaluation criterion is used to refer to how much resource fragmentation is located in VR-Cluster. We can calculate the sum of resource fragmentation in each function plane, as shown in the following formula:

$$\text{sum}(\text{frag}) = \sum_i^M \sum_p P_{C(m)} + \sum_j^N \sum_q P_{F(n)} + \sum_j^N \sum_q P_{S(e)} \quad (m \in C;\ n \in F;\ e \in S), \tag{11}$$

$P_{C(m)}$: it refers to resource fragmentation in control server m of control plane; $P_{F(n)}$: it refers to resource fragmentation in forwarding server n of forwarding plane; $P_{S(e)}$: it refers to resource fragmentation in interface server e of switching plane; $\sum_i^M \sum_p P_{C(m)}$: it refers to the magnitude of resource fragmentations in control plane; $\sum_j^N \sum_q P_{F(n)}$: it refers to the magnitude of resource fragmentations in forwarding plane $\sum_j^N \sum_q P_{S(e)}$: it refers to the magnitude of resource fragmentations in switching plane.

Magnitude of resource fragmentation is continually changing when operators frequently create and recycle router instances. If VR-Cluster has low magnitude of resource fragmentation, it has the higher chance to allocate physical resources for incoming requests of creation of new router instances; otherwise, it fails to establish new router instances, even though it has enough physical resources. Thus, magnitude of resource fragmentation is also an evaluation criterion to evaluate resource fragmentation problem in VR-Cluster.

4.2.3. Ratio of Resource Fragmentation. In order to further analyzing "resource fragmentation problem," we propose another evaluation criterion, namely, ratio of resource fragmentation. It refers to the ratio of the largest resource block to overall physical resource. The calculation of this evaluation criterion is as shown in the following formula:

$$f_R = \max\{f_C * \alpha, f_F * \beta, f_S * \chi\}. \tag{12}$$

In Formula (12), f_C refers to the ratio of resource fragmentation in control plane, as shown in Formula (13); f_F refers to ratio of resource fragmentation in forwarding plane, as shown in Formula (14); f_S refers to ratio of resource fragmentation in switching plane, as shown in Formula (15). In Formula (12), α, β, and γ are special parameters, whose value is just equal to 0 or 1.

When the ratio of resource fragmentation in control plane is not smaller than the ratio of resource fragmentation in forwarding plane and the ratio of resource fragmentation in switching plane, the ratio of resource fragmentation in VR-Cluster is determined by the ratio of resource fragmentation in control plane. And when both the ratio of resource fragmentation in control plane and the ratio of resource fragmentation in switching plane are smaller than the ratio of resource fragmentation in forwarding plane, the ratio of resource fragmentation in VR-Cluster is determined by the ratio of resource fragmentation in forwarding plane. f_R is always lower than 1; only when VR-Cluster does not establish any router instance, f_R is equal to 1.

Formula (13) is used to calculate the ratio of resource fragmentation in control plane: numerator refers to the largest

resource block in control plane and denominator refers to the total resources in control plane:

$$f_C = 1 - \frac{\max\left(\text{size}\left(P_{C(n)}\right)\right)}{C}. \quad (13)$$

Formula (14) is used to calculate the ratio of resource fragmentation in forwarding plane: numerator refers to the largest resource block in forwarding plane and denominator refers to the total resources in forwarding plane:

$$f_F = 1 - \frac{\max\left(\text{size}\left(P_{F(n)}\right)\right)}{F}. \quad (14)$$

Formula (15) is used to calculate the ratio of resource fragmentation in switching plane: numerator refers to the largest resource block in switching plane and denominator refers to the total resources in switching plane:

$$f_R = 1 - \frac{\max\left(\text{size}\left(P_{S(e)}\right)\right)}{S}. \quad (15)$$

5. VR-Mapping Algorithm

VR-mapping algorithm mainly calculates mapping relationship between router instances and substrate infrastructure. Besides, it tries to solve "resource fragmentation problem" by adjusting mapping relationship.

We introduce three algorithms into VR-Cluster: first-fit mapping algorithm, best-fit mapping algorithm, and worst-fit mapping algorithm. So we hope that these three algorithms also have ability to get an optimization mapping relationship between router instances and physical infrastructure and calculate an optimization adjustment scheme, which can efficiently solve "resource fragmentation problem" in VR-Cluster. And we mainly consider mapping relationship between LF-Planes and forwarding plane, because only LF-Plane can migrate from one to another when resource management calculate migration strategy.

5.1. First-Fit Mapping Algorithm. In first-fit mapping algorithm, it mainly includes two types of linked lists: (1) F-idle list used to manage idle resource blocks in forwarding plane and (2) F-allocated list used to manage forwarding resources occupied by router instances.

In first-fit mapping algorithm, it selects migrated router instance from the last to the first in F-allocated list and searches the first resource block in F-idle link including enough remaining resources for the migrated router instance. Once one router instance is moved, first-fit mapping algorithm determines whether or not VR-Cluster can allocate resources for new router instance: if not, first-fit mapping algorithm will dynamically move router instance again until VR-Cluster can establish new router instance or there is no unmoved router instance. So, in first-fit mapping algorithm, router instances are always located in the front servers in forwarding plane.

The processing of first-fit mapping algorithm is present in Algorithm 1. When resource management plane migrates one router instance, it allocates forwarding resources for the latter. Firstly, first-fit algorithm searches F-idle list to find the first resource block that meets requirements of LF-Plane (S2). At last, if it finds its selected resource blocks in F-idle list, it can allocate resources for this request (S3–S5); otherwise, it should dynamically move LF-Plane in VR-Cluster. For dynamic migration in first-fit algorithm (S6–S16), it selects the last router instances (S7) and searches the first resource block for it (S8–S13): if there is one resource block for migrated LF-Plane, it will move LF-Plane to this resource block (S9–S12). If it does not find wanted result (S14–S17), it will continue searching the first resource block for the last but one until it searches the last router instance or gets its wanted result (S6).

5.2. Best-Fit Mapping Algorithm. In best-fit mapping algorithm, it mainly includes two types of linked lists: (1) F-idle list used to manage idle resource blocks in forwarding plane and (2) F-allocated list used to manage forwarding resources occupied by router instances.

In best-fit mapping algorithm, it selects migrated router instance from the last to the first in F-allocated list and searches the first resource block including enough remaining resources for migrated router instance in F-idle list. Once one router instance is moved, best-fit mapping algorithm determines whether or not VR-Cluster can allocate for new router instance: if not, best-fit mapping algorithm will dynamically move router instance again until VR-Cluster can establish new router instance or there is no unmoved router instance. Best-fit mapping algorithm, compared with first-fit mapping algorithm, has to sort resource blocks from smallest to largest after VR-Cluster moves router instances or establishes new router instance. So, in best-fit mapping algorithm, router instances may be located in some different servers in forwarding plane.

The processing of best-fit mapping algorithm is present in Algorithm 2. When resource management plane establishes one router instance, it allocates forwarding resources for the latter. Firstly, best-fit algorithm searches F-idle list to find the first resource block that meets requirements of LF-Plane (S2). At last, if it finds its selected resource blocks in F-idle list, it can allocate resources for this request and sort F-idle list from smallest to largest (S3–S7); otherwise, it should dynamically move router instances in VR-Cluster. For dynamic migration in best-fit algorithm (S8–S23), it selects the last LF-Plane (S9) and searches the first resource block for it (S10–S16): if there is one resource block for migrated LF-Plane, it will move LF-Plane to selected resource block and sort F-idle list from smallest to largest (S11–S15). If it does not search wanted result (S17–S22), it will continue searching the first resource block for the last but one until it searches the last LF-Plane or gets its wanted result (S8).

5.3. Worst-Fit Algorithm. In worst-fit mapping algorithm, it mainly includes two types of linked lists: (1) F-idle list used to manage idle resource blocks in forwarding plane and (2) F-allocated list used to manage forwarding resources occupied by router instances.

In worst-fit mapping algorithm, it selects migrated router instance from the last to the first in F-allocated list and

```
Input: migration of router instance
Output: whether or not execute it
Process:
(1)   N = ∑ P_F(n); R = amount of VR;
(2)    while (n < N) {
(3)          if (P_F(n) ≥ R_F(n))
(4)                return SUCESS;
(5)    }
(6)    for (i = R; i ≤ R; i − −){
(7)          select Ri; //search the first resource block
(8)          while (n < N){
(9)               if (P_F(n) ≥ R^i_F(n)) {
(10)                    move Ri to P_F(n);
(11)                    Break;
(12)               }
(13)         }   //determine whether or not VR-Cluster meets
(14)         while (n < N){
(15)              if (P_F(n) ≥ R^x_F(n))
(16)                   return SUCESS;
(17)         }
(18)    }
```

Algorithm 1: First-fit mapping algorithm.

```
Input: migration of router instance
Output: whether or not execute it
Process:
(1)     N ∑ P_F(n); R = amount of VR;
(2)     while (n < N){
(3)          if (P_F(n) ≥ R^x_F(n)){
(4)               sort F-idle list from smallest to largest;
(5)          return SUCESS;
(6)       }
(7)     }
(8)   for (i = R; i ≤ R; i − −){
(9)        select Ri; //search the first resource block
(10)       while (n < N){
(11)            if (P_F(n) ≥ R^i_F(n)) {
(12)                 move Ri to P_F(n);
(13)                 sort F-idle list from smallest to largest;
(14)                 break;
(15)            }
(16)       }    //determine whether or not VR-Cluster meets
(17)       while (n < N){
(18)            if (P_F(n) ≥ R^x_F(n)){
(19)               sort F-idle list from smallest to largest;
(20)               return SUCESS;
(21)            }
(22)       }
(23)  }
```

Algorithm 2: Best-fit mapping algorithm.

searches the first resource block including enough remaining resources for migrated router instance in F-idle list. Once one router instance is moved, worst-fit mapping algorithm determines whether or not VR-Cluster can allocate new router instance: if not, best-fit mapping algorithm will dynamically move router instance again until VR-Cluster can establish new router instance or there is no unmoved router instance. Worst-fit mapping algorithm, compared with

```
Input: migration of router instance
Output: whether or not execute it
Process:
(1)   N = ∑ P_F(n); R = amount of VR;
(2)   while (n < N){
(3)         if (P_F(n) ≥ R^x_F(n)){
(4)               sort F-idle list from largest to smallest;
(5)               return SUCESS;
(6)         }
(7)   }
(8)   for (i = R; i ≤ R; i − −){
(9)       select Ri; //search the first resource block
(10)      while (n < N){
(11)          if (P_F(n) ≥ R^i_F(n)) {
(12)              move Ri to P_F(n);
(13)              sort F-idle list from largest to smallest;
(14)              break;
(15)           }
(16)      }   //determine whether or not VR-Cluster meets
(17)      while (n < N){
(18)          if (P_F(n) ≥ R^x_F(n)){
(19)              sort F-idle list from largest to smallest;
(20)              return SUCESS;
(21)          }
(22)      }
(23)  }
```

Algorithm 3: Worst-fit mapping algorithm.

first-fit mapping algorithm, has to sort resource blocks from largest to smallest after VR-Cluster moves router instances or establishes new router instance. So, in worst-fit mapping algorithm, router instances may be located in some different servers in forwarding resource pool and control resource pool.

The processing of worst-fit mapping algorithm is present in Algorithm 3. When resource management plane establishes one router instance, it allocates forwarding resources for the latter. Firstly, worst-fit algorithm searches F-idle list to find the first resource block that meets requirements of LF-Plane (S2). At last, if it finds its selected resource blocks both in F-idle-allocated list, it can allocate resources for this request and sort F-idle list from largest to smallest (S3–S7); otherwise, it should dynamically move LF-Plane in VR-Cluster. For dynamic migration in worst-fit algorithm (S8–S23), it selects the last LF-Plane (S9) and searches the first resource block for it (S10–S16): if there is one resource block for migrated LF-Plane, it will move LF-Plane to selected resource block and sort F-idle list from largest to smallest (S11–S15). If it does not search wanted result (S17–S22), it will continue searching the first resource block for the last but one until it searches the last LF-Plane or gets its wanted result (S8).

6. Experimental Results and Analysis

We firstly use general servers to achieve VR-Cluster including resource management plane, data plane, control plane, and switching plane. We then evaluate migration time based on VR-Cluster protoplatform. At last, we evaluate the amount of resource fragmentation, ratio of resource fragmentation, and failure magnitude of creation of router instances when VR-mapping algorithm uses different algorithms.

6.1. Experimental Environment. Our real implementation includes six servers and two switches, as shown in Figure 5: server with Intel Xeon CPU E5-2680 Processors, 64 GB RAM, dual 10 Gbps interfaces, and dual 10 Gbps interfaces, running Linux CentOS 6.5. One switch is 10 Gbps switch with 12 interfaces; another switch is 1 Gbps switch with 24 interfaces.

Control plane just consists of one server, which uses LXC to establish virtualization environment. The control interface interconnects with control fabric, so it can communicates with resource management plane and forwarding plane.

Resource management plane just includes one server. Its control interface interconnects with control fabric, which can communicate with control plane, forwarding plane, and switching plane.

Forwarding plane consists of two servers, running modified Click router. One 1 Gbps interface that is regarded as control interface interconnects with control fabric; one 10 Gbps interface that is regarded as data interface interconnects with data fabric.

Switching plane consists of two servers, running simple flow-table based application. One 1 Gbps interface that is regarded as control interface interconnects with control fabric; one 10 Gbps interface that is regarded as data interface interconnects with data fabric.

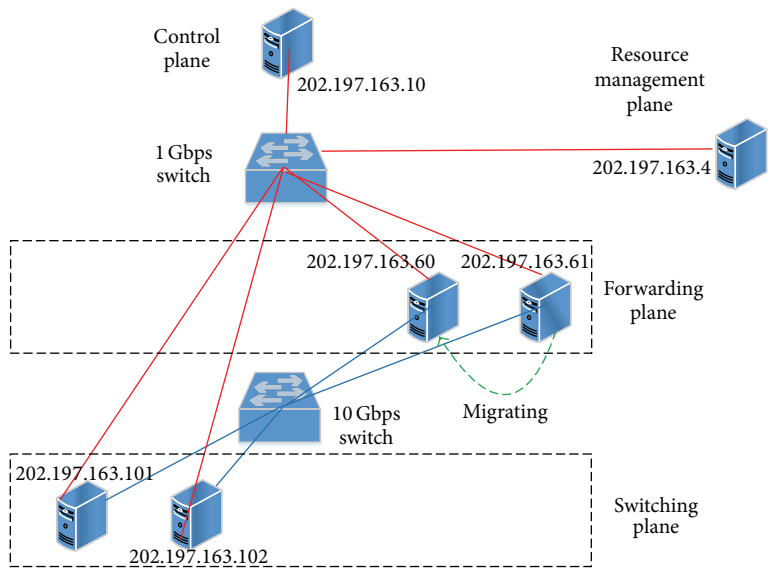


Figure 5: Experimental environment.

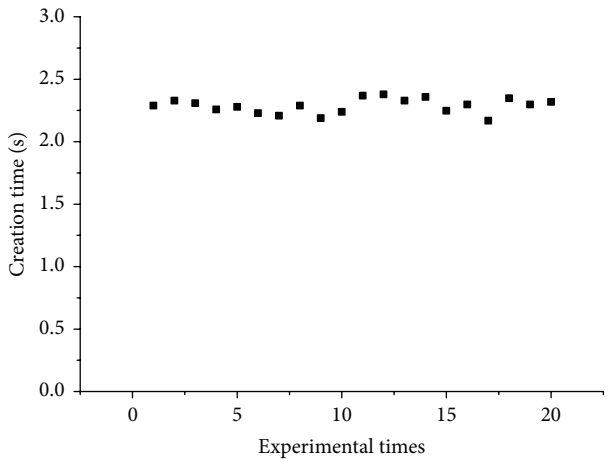


Figure 6: Creation time of LF-Plane.

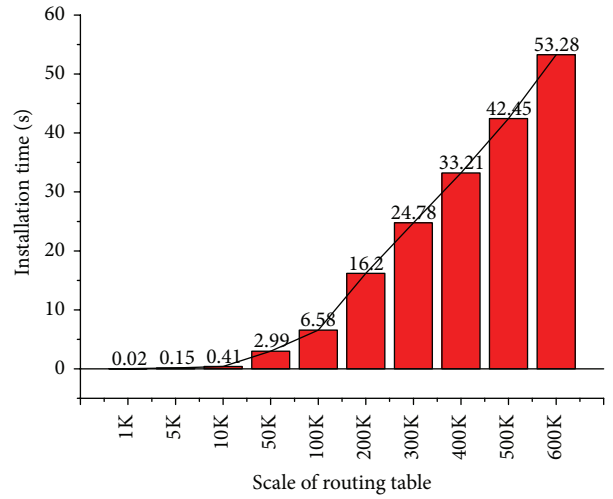


Figure 7: Installation time of routing table.

6.2. Migration Time. Migration time in VR-Cluster mainly includes two parts: (1) creation time of LF-Plane and (2) installation time of routing tables.

6.2.1. Creation Time of LF-Plane. In our VR-Cluster, LF-Plane adopts Click router to create LF-Plane. Besides, it uses DPDK tools to optimize I/O performance. When we migrate LF-Plane from one server to another server, we firstly create new LF-Plane in destination server. We evaluate twenty times of creation time of LF-Plane in forwarding plane, and our results are presented in Figure 6.

The average creation time of LF-Plane is about 2.3 s, because a new LF-Plane using DPDK tools has to take much time in initializing memory. And creation time is also affected by system loading in server of forwarding plane. If there are multiple LF-Planes, resource management plane uses more than 2.3 s to complete LF-Plane. Thus, creation time in VR-Cluster can meet our requirement, because old LF-Plane also continually forwards packets, which does not be beak off by creation of new LF-Plane.

6.2.2. Installation Time of Routing Table. In VR-Cluster, the time that LC-Plane takes to install a routing entry into its corresponding LF-Plane is about 0.3 ms. If the scale of routing table is 100 K, LC-Plane will takes about 55.4 s. In order to decrease installation time of routing table, we using packet batching technology to transmit multiple routing entries together. If MTU in control fabric is 1500, LC-Planes can install at most 121 routing entries once. We evaluate installation time of routing table under different scale of routing tables, as shown in Figure 7.

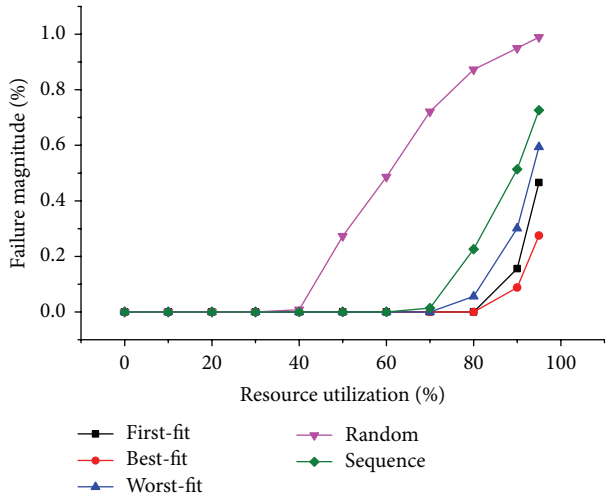


Figure 8: Failure magnitude.

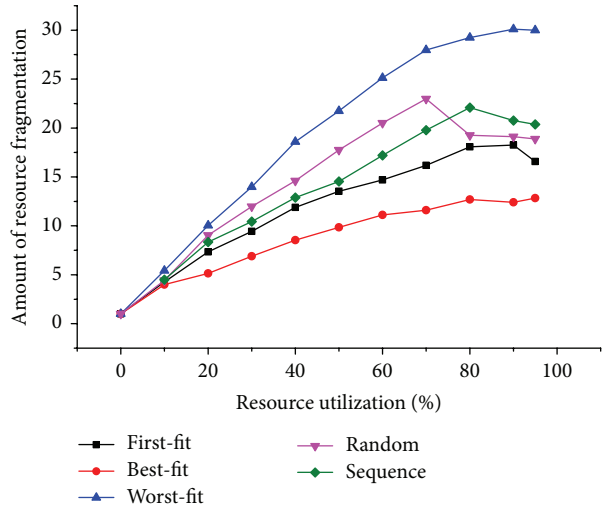


Figure 9: Amount of resource fragmentation.

In Figure 7, installation time continually grows with increase of scale of routing time. Relationship between scale of routing table and installation time does not exist linearly. Installation time it takes per time continually grows as the amount of routing entries it has installed increases, because LF-Plane has to take more time to search its routing tables when there is a huge amount of routing entries in it. For instance, LF-Plane takes about 0.56 ms to complete a single installation when it does not store any routing entry; when LF-Planes has stored 400 K routing entries, it may take about 1.09 ms.

6.2.3. Migration Time of LF-Plane. Migration time of LF-Plane is equal to the sum of creation time and installation time. When the scale of routing table is less than 10 K, migration time is mainly affected by creation time of LF-Plane. Once the scale of routing table is more than 50 K, migration time is mainly determined by creation time. In current Internet, a backbone router has about 500 K routing entries. In this context, migration time of LF-Plane is about 44.8 s, which is less than one minute. Thus, migration time of LF-Plane in VR-Cluster can meet our requirement.

6.3. Failure Magnitude of Creation of Router Instances. Failure magnitude of creation of router instances can be reduced by dynamic migration in VR-Cluster. In this part, we use it to evaluate whether or not dynamic migration is useful and to determine whether or not efficiency of dynamic migration is related to VR-mapping algorithms, as shown in Figure 8.

It shows that failure magnitude of creation of router instances increases as resource utilization continually increases. Random mapping algorithm firstly shows failure of creation of router instances when resource utilization is about 40%. And it has a worse performance as resource utilization continually increases. Sequence mapping algorithm is better than random mapping algorithm, because the latter may result in lots of small resource fragmentation. When resource utilization is too high, these two algorithms have a bad performance. For example, failure magnitude of creation of router instances in random mapping algorithm and sequence mapping algorithm is correspondingly 98.9% and 72.6% at 95% resource utilization. In our proposed three dynamic mapping algorithms, they show the first failure of creation of router instances only when the resource utilization is higher than 80%. Besides, performance of our proposed algorithms is better than the other two algorithms. In particular, best-fit mapping algorithm just has 27.5% failure magnitude when resource utilization is 95%. So, best-fit mapping algorithm is better than other proposed algorithms in terms of failure magnitude of creation of router instances.

6.4. Amount of Resource Fragmentation. Amount of resource fragmentation is another useful criterion to evaluate efficiency of dynamic migration. In this part, we mainly measure the amount of resource fragmentation generated in processing of creation and deletion of router instances, as shown in Figure 9.

Worst-fit mapping algorithm generates more resource fragmentation than other algorithms, because this algorithm usually allocates the largest resource fragmentation for router instances. Most of algorithms may generate more resource fragmentation as resource utilization continually increases. However, there is a special phenomenon in three algorithms including first-fit, random, and sequence mapping algorithm: the amount of resource fragmentation decreases after the resource utilization exceeds a defined value. Its reason is that VR-Cluster has a higher failure magnitude of creation of router instances (as shown in Figure 8) when resource utilization exceeds a defined value. However, the amount of resource fragmentation in best-fit mapping algorithm does not decrease, because it has a lower failure magnitude of creation of router instances. Best-fit mapping algorithm is also better than other algorithms, which always has no more than fifteen kinds of resource fragmentation.

6.5. Ratio of Resource Fragmentation. We measure the maximum size of resource fragmentation and calculate ratio of

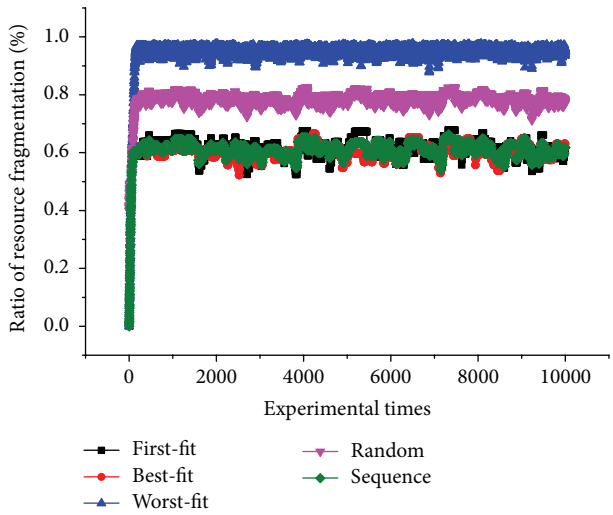


Figure 10: Amount of resource fragmentation.

resource fragmentation when resource utilization is 50%, as shown in Figure 10. When resource utilization is 50%, these algorithms except for random mapping algorithm do not show failure magnitude of creation of router instances.

It shows that ratio of resource fragmentation increases severely with a start and stays in a fixed position. The fixed value in worst-fit mapping algorithm is approximately 92%; the fixed value in random mapping algorithm is about 78%; and the fixed value of first-fit mapping algorithm is 60%, which is the same as the fixed value in best-fit and sequence mapping algorithm. The reason is that worst-fit mapping algorithm always allocates the maximal resources for router instances, and random mapping algorithm also potentially splits big resource fragmentation into small pieces of resource fragmentation. From aspect of ratio of resource fragmentation, first-fit algorithm and best-fit algorithm are better than worst-fit algorithm.

6.6. Summary. From the above experimental results, migration time in VR-Cluster can be accepted, which cannot result in downtime of router instances. Thus, VR-Cluster can use dynamic migration to solve "resource fragmentation problem."

We further explore efficiency of our proposed three VR-mapping algorithms. These three algorithms are better than random and sequence mapping algorithm. And best-fit mapping algorithm is the best in five mapping algorithms. It can reduce failure magnitude of creation of router instances, amount of resource fragmentation, and ratio of resource fragmentation.

7. Conclusion and Future Works

We find that there are lots of resource fragmentation in the processing of creation and deletion of router instances. And we firstly put forward "resource fragmentation problem" and establish evaluation model to analyze the above problem. In order to prove efficiency of dynamic migration, we establish a virtual router platform, called VR-Cluster, and use it to measure migration time of router instances. At the same time, this paper further proposes three VR-mapping algorithms including first-fit mapping algorithm, best-fit mapping algorithm, and worst-fit mapping algorithm. At last, experimental results prove that migration time of router instance can meet requirements of dynamical migration. Besides, best-fit mapping algorithm is the best to solve "resource fragmentation problem" in five algorithms.

In next works, we will explore a better VR-mapping algorithm than best-fit mapping algorithm, which can consider green energy, system loading balance, and so forth.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is supported by Program for National Basic Research Program of China (973 Program) "Reconfigurable Network Emulation Testbed for Basic Network Communication" and research on XXX access authentication and authorization protocol standards.

References

[1] N. M. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[2] D. Unnikrishnan, R. Vadlamani, Y. Liao et al., "Scalable network virtualization using FPGAs," in *Proceedings of the 18th ACM SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '10)*, pp. 219–228, Monterey, Calif, USA, February 2010.

[3] F.-E. Zaheer, J. Xiao, and R. Boutaba, "Multi-provider service negotiation and contracting in network virtualization," in *Proceedings of the 12th IEEE Network Operations and Management Symposium (NOMS '10)*, pp. 471–478, IEEE, Osaka, Japan, April 2010.

[4] D. Cotroneo, L. De Simone, A. K. Iannillo et al., "Network function virtualization: challenges and directions for reliability assurance," in *Proceedings of the 25th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW '14)*, pp. 37–42, IEEE, Naples, Italy, November 2014.

[5] R. Mijumbi, J. Serrat, J. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, 2016.

[6] S. Bhatia, M. Motiwala, W. Muhlbauer et al., "Hosting virtual networks on commodity hardware," Georgia Tech Computer Science Technical Report GT-CS-07-10, 2008.

[7] J. S. Urner, P. Crowley, J. DeHart et al., "Supercharging planetlab: a high performance, multi-application, overlay network platform," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 85–96, 2007.

[8] M. B. Anwer and N. Feamster, "Building a fast, virtualized data plane with programmable hardware," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 75–82, 2010.

[9] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Fairness issues in software virtual routers," in

Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO '08), pp. 33–38, ACM, Seattle, Wash, USA, August 2008.

[10] G. Xie, P. He, H. Guan et al., "PEARL: a programmable virtual router platform," *IEEE Communications Magazine*, vol. 49, no. 7, pp. 71–77, 2011.

[11] V. Eramo, E. Miucci, and M. Ammar, "Study of migration policies in energy-aware virtual router networks," *IEEE Communications Letters*, vol. 18, no. 11, pp. 1919–1922, 2014.

[12] X. Gao, B. Wang, X. Zhang et al., "Evaluation and analysis of three typical resource allocation algorithms in virtual router platform," in *Algorithms and Architectures for Parallel Processing: 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18–20, 2015, Proceedings, Part IV*, vol. 9531 of *Lecture Notes in Computer Science*, pp. 549–568, Springer, Berlin, Germany, 2015.

[13] X. Chen and C. Phillips, "Virtual router migration and infrastructure sleeping for energy management of IP over WDM networks," in *Proceedings of the International Conference on Telecommunications and Multimedia (TEMU '12)*, pp. 31–36, Chania, Greece, August 2012.

[14] M. L. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[15] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 231–242, 2008.

[16] Y. Wang, J. V. D. Merwe, and J. Rexford, "VROOM: virtual routers on the move," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 1–7, 2008.

[17] M. Dowty and J. Sugerman, "GPU virtualization on VMware's hosted I/O architecture," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, pp. 73–82, 2009.

[18] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[19] D. Bernstein, "Containers and cloud: from LXC to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[20] N. Pitropakis, D. Anastasopoulou, A. Pikrakis, and C. Lambrinoudakis, "If you want to know about a hunter, study his prey: detection of network based attacks on KVM based cloud environments," *Journal of Cloud Computing*, vol. 3, no. 1, pp. 1–10, 2014.

[21] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador et al., "QuagFlow: partnering quagga with openflow," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 441–442, 2010.

[22] M. Handley, O. Hodson, and E. Kohler, "XORP: an open platform for network research," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 53–57, 2002.

[23] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[24] Y. Li and G. Wang, "SDN-based switch implementation on network processors," *Communications and Network*, vol. 5, no. 3, pp. 434–437, 2013.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 9524379, 9 pages
http://dx.doi.org/10.1155/2016/9524379

Research Article

FP-ABC: Fast and Parallel ABC Based Energy-Efficiency Live VM Allocation Policy in Data Centers

Jianhua Jiang,[1,2,3] Yunzhao Feng,[1,3] Milan Parmar,[4] and Keqin Li[5]

[1] *School of Management Science and Information Engineering, Jilin University of Finance and Economics, Changchun 130117, China*
[2] *Laboratory of Logistics Industry Economy and Intelligent Logistics, Jilin University of Finance and Economics, Changchun 130117, China*
[3] *Jilin Province Key Laboratory of Internet Finance, Jilin University of Finance and Economics, Changchun 130117, China*
[4] *International Department, Faculty of Computer Science and Information Management, Hunan University of Arts and Science, Changde, Hunan 415000, China*
[5] *Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*

Correspondence should be addressed to Keqin Li; lik@newpaltz.edu

Received 23 March 2016; Accepted 16 June 2016

Academic Editor: Fabrizio Riguzzi

Virtual machine (VM) technology is one of the energy-efficiency approaches to save energy with acceptable quality of service (QoS). In our previous studies, Artificial Bee Colony (ABC) based VM allocation policy can make a good tradeoff between performance and energy consumption. However, there are two problems in state-of-the-art ABC based approaches: (1) how to find global optimized solutions efficiently; (2) how to minimize the decision time of VM allocation. To solve these two problems, the idea of simulated annealing is adopted to get a better global optimum, and the idea of gradient descent is applied to accelerate the speed of finding solution space in Δt. Compared with state-of-the-art ABC based policies, the experimental results show that the proposed algorithm efficiently reduces energy consumption and SLA violation.

1. Introduction

At present, a large number of data centers have been put into practice to support internet-based services. Many applications, such as E-commerce and scientific computing, require large-scale data-intensive computing. As data centers and their applications continue increasing, how to realize energy-efficiency computing has become a particular challenge.

There are several properties of data centers making the problem of energy-efficiency computing difficult to be solved. First, workload of a data center is a priori unknown to energy-efficiency policy and will likely be variable over both time and space. As a result, a static energy-efficiency policy is insufficient. Second, customers wish their jobs can be finished in time with an acceptable price. Third, lots of computing and storage nodes are occupied in finishing their jobs in time with high energy consumption and low utilization. Therefore, finding a tradeoff between energy consumption and performance in data centers is an important issue.

Many practices have been applied to achieve energy-efficiency, such as Dynamic Voltage and Frequency Scaling (DVFS) [1] and virtual machine technique [2]. Virtual machine technique can improve the utilization rate of resources, which are computing and storage nodes. For example, low-load nodes will be slept to save energy when their VMs have been migrated to other nodes with workload balance principles.

VM allocation policies with workload balance are considered as one of the important energy-efficiency solutions in data centers. In CloudSim [3] simulator, the classical policies have been proposed, such as Random Selection (RS), Minimum Migration Time (MMT), and Median Absolute Deviation (MAD), to make VM allocation decisions. Zhao et al. [4] applied PSO algorithm to VM allocation

policy, and they have achieved significantly on the measurement of energy-efficiency and QoS. In our previous works, Artificial Bee Colony (ABC) [5] is adopted to achieve energy-efficiency [6]. However, ABC based VM allocation policies have disadvantages of finding global solutions in time.

To solve these problems, the idea of simulated annealing is adopted to find the global solution as much as possible in this paper. Based on the principle of simulated annealing, following bees have probability to jump out of local optimum to find global optimum in other solution space. Even more, the idea of gradient descent is applied to accelerate the speed of finding solution in Δt with QoS. Considering these two ideas, an ABC based energy-efficiency live VM allocation policy with simulated annealing and gradient descent is proposed in this paper.

The main contributions of the paper are summarized as follows:

(i) The idea of gradient descent is used to solve the problem that is how to satisfy the requirement of accelerating the speed of finding local optimum solution in a data center.

(ii) The idea of simulated annealing is adopted to get m approximate global optimums with searching in turn per Δt duration.

(iii) After finding global optimums in time, the frequency of VM migrations is reduced greatly to achieve a fairly high energy-efficiency level.

The rest of this paper is organized as follows. Related work is depicted in Section 2. The proposed VM allocation policy is given in Section 3. Simulated experiments are done in Section 4. Discussion and analysis are given in Section 5. Section 6 concludes the paper.

2. Related Work

The virtual machine allocation (VM allocation) is a process that is finding the target host for the live migrant VM. Traditionally, VM allocation policy is designed to achieve high performance. Currently, energy consumption is considered as another important influencing factor to evaluate the reasonability of VM allocation policy. Therefore, performance and energy consumption are two major influencing factors to construct energy-efficiency evaluation model.

Fan et al. [7] found a strong correlation between single server and CPU utilization rate; that is, there is a direct linear correlation between CPU utilization rate and the server's energy consumption. The higher the CPU utilization rate is, the more the energy consumption is. This is represented by a host energy evaluation model:

$$P_{\text{total}} = P_{\text{idle}} + \left(P_{\text{busy}} - P_{\text{idle}}\right)\mu_{\text{cpu}}, \tag{1}$$

where P_{total} represents the total energy consumption of a host, P_{idle} represents the energy consumption of an idle host, P_{busy} represents the energy consumption of a high load host, and μ_{cpu} indicates the CPU utilization rate of a host. But it cannot measure the nonlinear CPU utilization rate.

Boglazov [8] proposed another empirical nonlinear model to evaluate the energy consumption of a host:

$$P_{\text{total}} = P_{\text{idle}} + \left(P_{\text{busy}} - P_{\text{idle}}\right)\left(2\mu_{\text{cpu}} - \mu_{\text{cpu}}^{\gamma}\right), \tag{2}$$

where γ represents calibration parameters for minimizing the squared error, which is obtained by experiments.

For the cost of energy consumption from data centers, researchers have found a variety of efficient policies to reduce the cost of energy and keep a relatively high QoS level.

Deng et al. [9] proposed the concept of green data centers, which means low energy consumption, low cost, and low levels of environment pollution. Fiandrino et al. [10] proposed the model of energy consumption for VM migration, which can reduce the cost of energy consumption of data centers to achieve workload balance. However, they did not discuss the problem of finding global optimum and unsuitable VM migration. Nakada and Hirofuchi [11] optimized the number of physical hosts and reduced the frequency of virtual machine migrations. They made Service Level Agreement (SLA) as an important influencing factor to measure the optimal results and also placed VM migration as a multiobjective optimization issue.

Nishant et al. [12] utilized the ant colony algorithm to solve the problem of VM migration. Each ant only updates its local data, which leads to the low rate of convergence. Jeyarani et al. [13] presented SAPSO to realize dynamic VM scheduling in data centers. SAPSO promptly detects and efficiently tracks the changing optimum that represents target servers for VM placement. However, the idea did not take future workload into account. From the perspective of long operation, the policy may incur additional costs.

Previously, our research focuses on making use of ABC algorithm to solve the problem of VM allocation, but it is insufficient on the speed of convergence for finding m local optimums in data centers. When dealing with the same volume of jobs, the higher the energy-efficiency level of target host is, the less it costs. As we all know, the energy-efficiency level will be decreased rapidly when the workload of a target host is larger than a certain critical value. Therefore, it is important to achieve a fairly high energy-efficiency level in data centers by making an effective live VM allocation policy. In our previous researches, ABC based energy-efficiency live VM allocation policy has been proposed. However, there are still two problems which need to be solved in Section 1.

3. ABC Based Live VM Allocation Policy with Simulated Annealing and Gradient Descent

Workload balance in data centers can save energy by migrating VMs to target hosts, and the key is to assign VM to the optimal target host [14]. To achieve the goal of saving energy in data centers, we introduce an ABC algorithm to seek the best target hosts. However, Bi and Wang [15] found that an ABC algorithm is easy to fall into local optimum. To solve this problem, simulated annealing makes the scouts jump out

of local optimal solution space and then optimize the optimal solution within multiple iterations. The following paper will elaborate it from two aspects, one is the energy evaluation model of data centers, and the other is an enhanced energy-efficiency live VM allocation policy.

3.1. Data Center Energy Evaluation Model. In the live VM allocation process, we want to assess the cost that the VM needs to be migrated to the target host. We use P_{diff}, which represents difference in the energy consumption, to evaluate virtual machine migration process cost:

$$\begin{aligned} P_{\text{after}} &= P_{\text{idle}} + \left(P_{\text{busy}} - P_{\text{idle}}\right)\left(2\mu_{\text{cpu\_after}} - \mu^{\gamma}_{\text{cpu\_after}}\right) \\ P_{\text{now}} &= P_{\text{idle}} + \left(P_{\text{busy}} - P_{\text{idle}}\right)\left(2\mu_{\text{cpu\_now}} - \mu^{\gamma}_{\text{cpu\_now}}\right) \\ P_{\text{diff}} &= 2\left(P_{\text{busy}} - P_{\text{idle}}\right)\left(\mu_{\text{cpu\_after}} - \mu_{\text{cpu\_now}}\right) \\ &\quad + \left(P_{\text{busy}} - P_{\text{idle}}\right)\left(\mu^{\gamma}_{\text{cpu\_now}} - \mu^{\gamma}_{\text{cpu\_after}}\right), \end{aligned} \tag{3}$$

where P_{after} corresponds to the energy consumption of a target host after receiving VM and P_{now} means the current energy consumption of the target host.

On the basis of the energy evaluation model, the metric of fitness is defined as in

$$F_{\sigma,\omega} = \tan\theta = \frac{E_\sigma - E_\omega}{\sqrt{\left(x_\sigma - x_\omega\right)^2 + \left(y_\sigma - y_\omega\right)^2}}, \tag{4}$$

where E corresponds to the energy consumption and σ and ω mean two different physical hosts.

3.2. Formal Problem. We formalized the problem as migrating n virtual machines to m hosts. Its solution is represented as n-dimensional vector, where each element represents a target host. We can describe the problem as to find a virtual machine allocation policy; we define it as A, which is to satisfy the demand to maximize computing performance while minimizing energy consumption. To solve this problem, we define an array of $S = \{H, \text{VM}, E, A\}$. H represents m physical hosts; $H(m,t) = \{H_1, H_2, H_3, \ldots, H_m\}$, where t means the VM migration time and VM represents n VMs. $\text{VM}(n,t,\Delta t) = \{\text{VM}_1, \text{VM}_2, \ldots, \text{VM}_n\}$ is expressed as n VM migration demand in Δt. $E(m,t) = \{E_1, E_2, E_3, \ldots, E_m\}$ represents the total energy consumption of m hosts. The purpose of the algorithm is to find the host location for maximum performance and minimize energy consumption. There are multiple locations to meet the demand, so we define as (6), $i \in \{1,2,3,\ldots,n\}$, $j \in \{1,2,3,\ldots,m\}$, $r \in \{1,2,3,\ldots,s\}$, φ represents a different location selected that is migrating VM to number j host, and it has the following relationship:

$$\varphi = \begin{cases} 1 & \text{if VM has been allocated to } H_j \text{ and } rcr\text{VM} \le acrH_j \\ 0 & \text{if VM has been allocated to } H_j \text{ and } rcr\text{VM} \ge acrH_j \\ \text{Invalid} & \text{if VM has not been allocated to } H_j, \end{cases} \tag{5}$$

$$Z_s = \sum_{i=1}^{n}\sum_{j=1}^{m}\varphi, \tag{6}$$

$$\Delta E = \delta^r_k - \delta^r_{k-1}, \tag{7}$$

$$\delta E = \sum_{k=1}^{m}\left(\delta^r_k - \delta^r_{k-1}\right), \tag{8}$$

where rcrVM represents the smallest computing nodes of VMs and $acrH_j$ marks the available computing resources of jth host. There are o positions for max φ that can be selected in S placement policy. It can be expressed as $\text{PS} = \{m, \text{VM}, o, t\}$.

The second parameter is based on energy consumption; $A = \{m, \text{VM}, r, t + t_k\}$, where r corresponds to any placement of o and k represents the host number. The position o, when selecting the host number k, is represented as $A = \{m, \text{VM}, r, t + t_k\}$. Their energy is represented as δ^{γ}_k, and it means, according to r, the total energy consumption that is migrating VMs to the host. Hence migrating VM energy consumption can be expressed as (7). In order to achieve better energy-efficiency, energy-efficiency VM migration can be represented as (8). In other words, in order to minimize energy consumption, we must minimize the value of δE.

3.3. Expressed Solutions. In order to achieve an ABC based energy-efficiency algorithm, representation of the solution becomes a priority. Solutions expressed represent the correspondence between the optimal bee and problem spaces found by ABC based algorithm. In this paper, there are nVMs to be migrated to m target hosts, so this is an n-dimensional vector problem. Each dimension is an integer

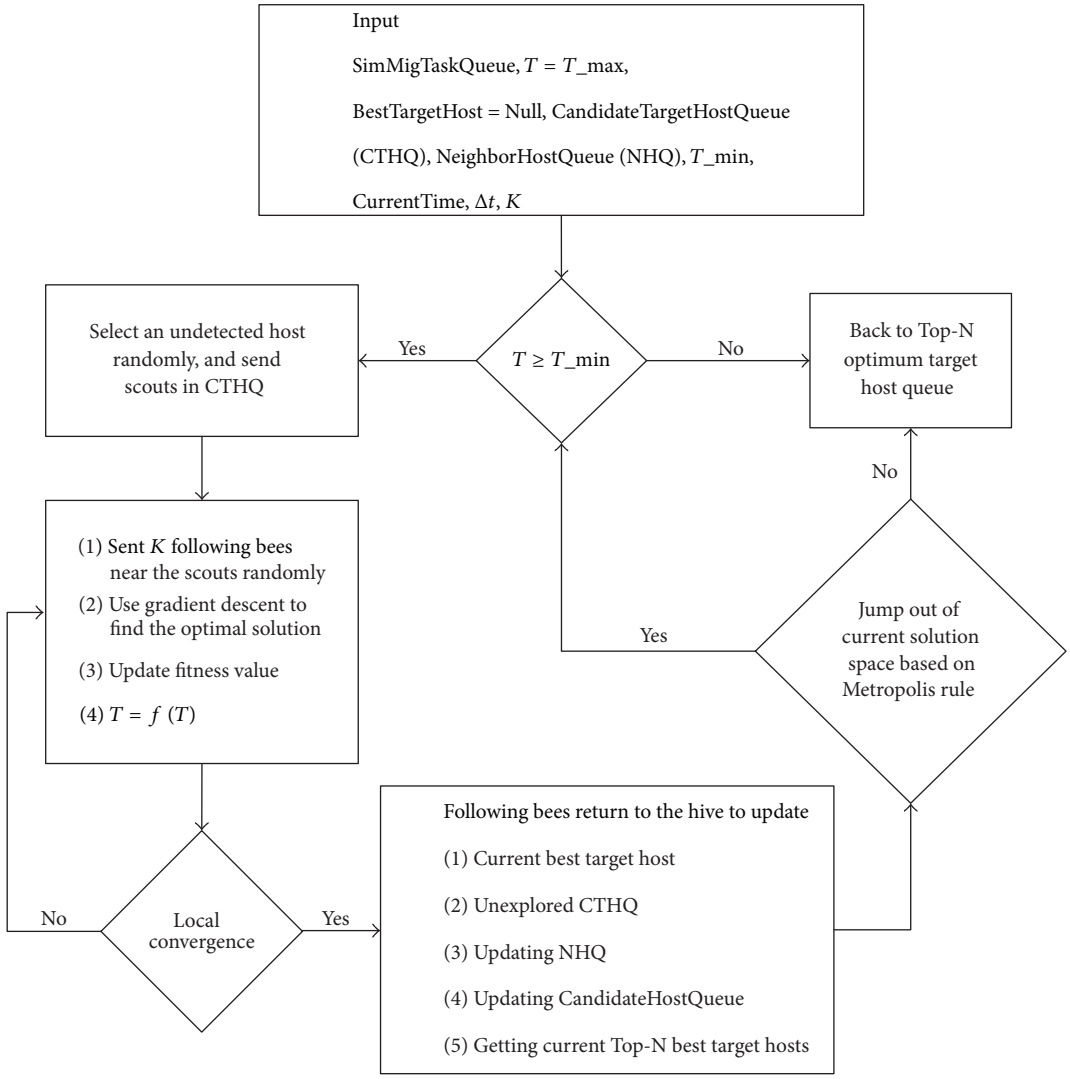


Figure 1: Flowchart of FP-ABC VM allocation policy.

from 0 to $m - 1$. Position vector is represented as $gBest_i = \{gB_{i1}, gB_{i2}, \ldots, gB_{iD}\}$. v_{ij} shows a nectar of the ith (The ith solution vector) position of the jth (jth VMs to be migrated) dimension. Velocity vector is represented as $v_i = \{v_{i1}, v_{i2}, \ldots, v_{ij}, \ldots, v_{iD}\}$; v_{ij} shows j-dimensional speed of the ith nectar source. $pBest_i = \{pB_{i1}, pB_{i2}, \ldots, pB_{iD}\}$ shows that the current optimal location is found, and best position vector is represented as $gBest_i = \{gB_{i1}, gB_{i2}, \ldots, gB_{iD}\}$, which is regarded as the current best target location in the population.

3.4. The Main Idea of ABC Based Energy-Efficiency Live VM Allocation Policy. An ABC based live VM allocation policy has been considered as a better energy-efficiency policy. According to [15], it can reduce 25%~30% energy cost with acceptable SLA violation. However, in big data time, the ABC based energy-efficiency live VM allocation policy will face huge amount of data-intensive jobs and need to get the best VM allocation decision in time.

To solve these problems, we have improved the ABC algorithm in two ways. First, the traditional speed of ABC based algorithm uses fitness function to find local optima which need to be optimized. Therefore, the idea of gradient descent is applied to getting the local optima rapidly. Second, the traditional ABC based algorithm maybe falls into the local optima in a time window of Δt. Therefore, simulated annealing is applied to finding the global optima.

3.5. Improved Artificial Bees Live VM Allocation Policy. As illustrated in Figure 1, the FP-ABC VM allocation policy can be described as shown in Algorithm 1.

4. Results

4.1. Experimental Environment. The cloud simulator of CloudSim3.0 [3] is used to conduct a simulation experiment. Java programming language is adopted to get results from

Input: the candidate target host queue CTHQ = $\{h_1, h_2, \ldots, h_n\}$
Output: *BestTargetHost*
Pseudo-code as follows:
(1) CTHQ = $\{h_1, h_2, \ldots, h_n\}$ are marked unexplored; Set the *SimMigTaskQueue*;
(2) Set the maximum temperature *(T*_max*)*, T = T_max and T_min;
(3) Initialize *CandidateTargetHostQueue*;
(4) Set *NeighborHostQueue* to be *Null*, and set *BestTargetHost* also to be *Null*;
(5) Set the number of following bees as K;
(6) Set the initial time point t, // set the number of n VM migration requests;
(7) **While** ($T \geq T$_min)
(8) Select an undetected host randomly, and send scouts in candidate target host queue;
(9) do
(10) **Parbegin**
(12) Calculates the host power // Sent K following bees near the scouts randomly
(13) Use of gradient descent, calculation and evaluation function

$$J(E) = \frac{1}{2}\sum_{i=1}^{m}\left(E\left(x_i, y_i\right)\right)^2$$

(14) **If** ($J(E) = 0$)
(15) Output the host number of current nearby following bee // find current local optima
(16) **else** {
(17) in accordance with the rules of

$$\min f\left(x_i + \alpha\frac{\partial}{\partial x_i}E\left(x_i, y_j\right)\right), \min f\left(y_j + \alpha\frac{\partial}{\partial x_i}E\left(x_i, y_j\right)\right)$$

to apply the program of the gradient descent
(18) $x_{i+1} = x_i - \alpha\frac{\partial}{\partial x_i}E\left(x_i, y_j\right),\ y_{j+1} = y_j - \alpha\frac{\partial}{\partial y_j}E\left(x_i, y_j\right)$
}
(19) **Endif**
(20) use

$$E\left(x_i - \alpha\frac{\partial}{\partial x_i}E\right) = \min_{\alpha>0} E\left(x_i - \alpha\frac{\partial}{\partial x_i}E\right) \text{ to update } \alpha_k;$$

(21) **ParEnd**
(22) $T_k = T_0/k^m$ // Make use of simulated annealing.
(23) Update fitness value,

$$F_{i,j} = \tan\theta = \frac{E_\sigma - E_\omega}{\sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}};$$

(24) **While** ($F_{i,j} = 0$)
(25) Following bees have returned, update current best target host;
(26) Update unexplored candidate target host group;
(27) Update the nearby hosts, candidate host queue and *Top-N* current best target host queue
(28) Recording time t;
(29) **Endwhile**
(30) $\Delta E = E - E0$ // Out of the current local optimal solution based on the Metropolis criterion
(31) **If** ($\exp(-\Delta E/T) >$ random(0, 1))
(32) Return to *Line (7)*
(33) **else**
(34) Output BestTargetHost
(35) **Endwhile**
(36) Output similar target hosts

Algorithm 1

MyEclipse9.0. The experiment is done on Lenovo™ desktop. Its system parameters as follows:

CPU: Intel(R) Core(™) i7-3770 CPU @ 3.40 GHz.

RAM: 4.00 GB.

System type: 64-bit operating system.

OS: Windows 7.

PlanetLab [8] is a group of computers, which is tested for computer networking and distributed systems research. It was established in 2002 by Professor Larry L. Peterson and Professor David Culler, and it was composed of 1353 nodes at 717 sites worldwide in January 2016 [16]. 10-day sample data in PlantLab cloud computing environment are chosen as experimental data as shown in Table 1 [8]. Simulation data center is

Table 1: Characteristics of the workload data (CPU utilization rate).

Date	The number of VMs	Mean	Standard deviation	Quartile 1	Median	Quartile 3
03/03/2011	1052	12.31%	17.09%	2%	6%	15%
06/03/2011	898	11.44%	16.83%	2%	5%	13%
09/03/2011	1061	10.70%	15.57%	2%	4%	13%
22/03/2011	1516	9.26%	12.78%	2%	5%	12%
25/03/2011	1078	10.56%	14.14%	2%	6%	14%
03/04/2011	1463	12.39%	16.55%	2%	6%	17%
09/04/2011	1358	11.12%	15.09%	2%	6%	15%
11/04/2011	1233	11.56%	15.07%	2%	6%	16%
12/04/2011	1054	11.54%	15.15%	2%	6%	16%
20/04/2011	1033	10.43%	15.21%	2%	4%	12%

Table 2: Power consumption by the selected servers at different workload levels in Watts.

Workload level	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP ProLiant G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP ProLiant G4	93.7	97	101	105	110	116	121	125	129	133	135

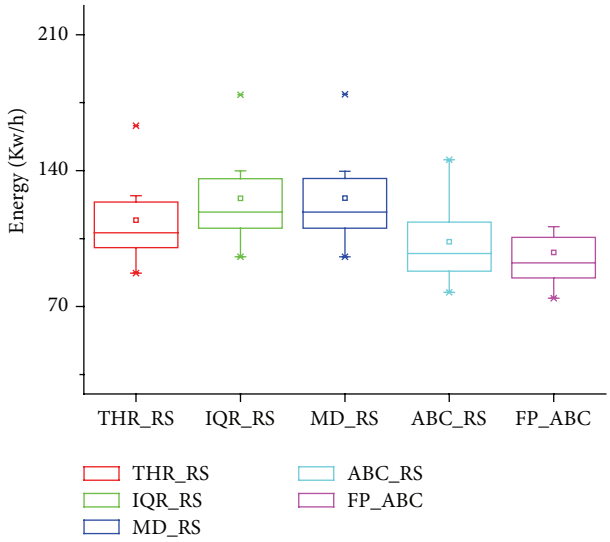


Figure 2: Energy consumption of the five algorithms.

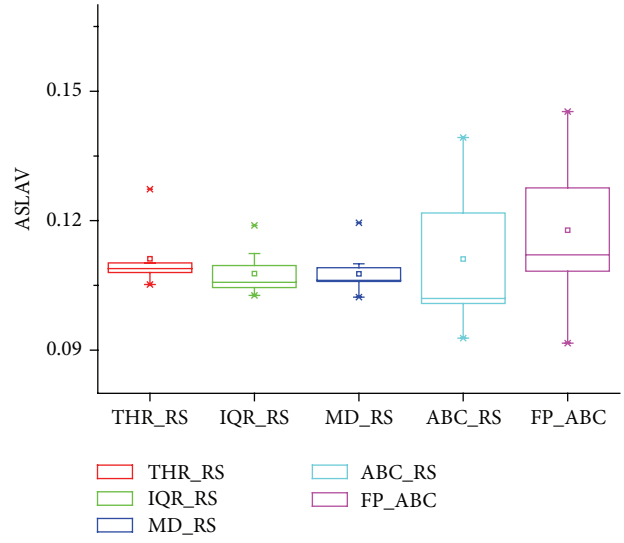


Figure 3: ASLAV metric of the five algorithms.

composed of 800 heterogeneous physical nodes, and half of these nodes are HP ProLiant ML100 G4 server (1860 MIPS) while the other halves are HP ProLiant ML110 G5 server (2660 MIPS) [8]. Energy consumption of these two servers at different loads is shown in Table 2. Server virtual machine is set as single-core; RAM can also be classified according to the number of VMs. The following are the main types of VMs: High-CPU Medium Instance (2500 MIPS, 0.85 GB), Extra Large Instance (2000 MIPS, 3.75 GB), Small Instance (1000 MIPS, 1.7 GB), and Micro Instance (500 MIPS, 613 MB).

The simulation is performed in CloudSim3.0, Δt is set as 30 seconds, and its safety parameter is 1.5.

4.2. Evaluating the Energy Consumption of FP-ABC with Different Approaches. In Figure 2, FP-ABC can reduce the energy consumption of data center by 5.1% on the base of ABC_RS. The proposed energy evaluation model can be used to evaluate different VM allocation policies. Our proposed FP-ABC policy has advantage in saving energy.

4.3. Evaluating the Violation Rate of SLA with Different Approaches. Service Level Agreement (SLA) represents a guarantee under certain overload performance and reliability of service. In other words the service provider and the user-defined agreements are accepted by each other. SLAV is short for Violations of SLA. ASLAV is the average rate of SLAV. The QoS will be better if its ASLAV is lower. In Figure 3, FP-ABC is the last one. Among these traditional policies, THR_RS has the worst performance. And it proves that VM allocation policy of THR_RS gets higher ASLAV value. The

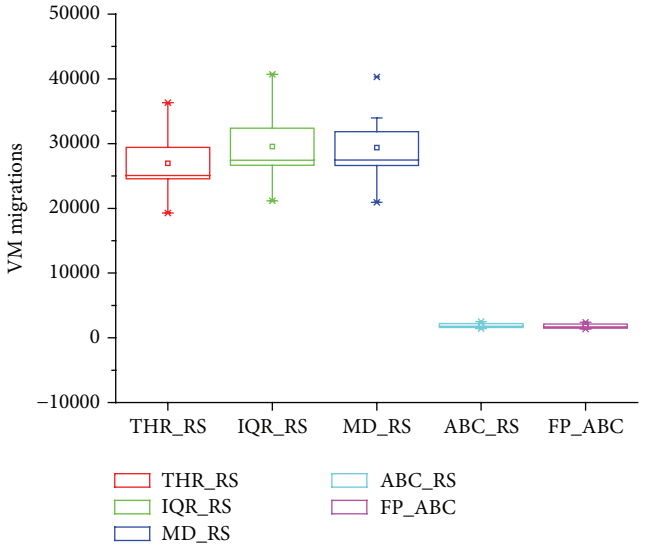


Figure 4: Number of VM migrations of the five algorithms.

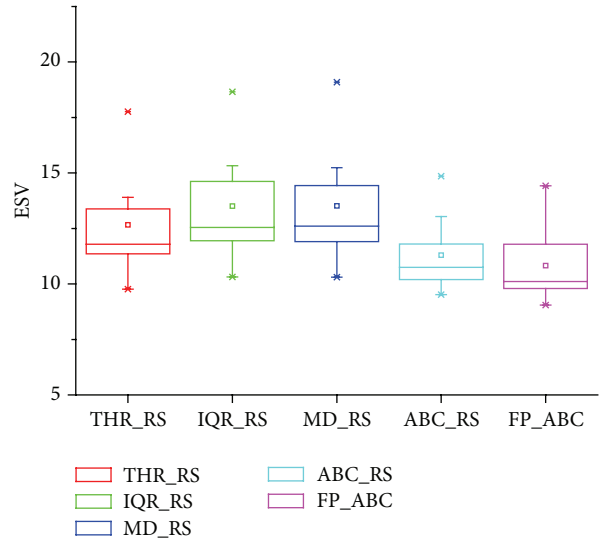


Figure 5: ESV metric of the five algorithms.

ASLAV value of FP-ABC is 10.4%, and it means that FP-ABC can reduce energy consumption with acceptable ASLAV. Therefore, the workload balance based policy can satisfy user's requirement of QoS.

4.4. Evaluating the Effectiveness of FP-ABC with Different VM Allocation Policies. The number of VM migrations is a statistic variable to measure the VM migration times which occurred in data centers. The cost will be lower if fewer VM migrations occurred. As shown in Figure 4, FP-ABC and ABC_RS can reduce the frequency of VM migrations greatly. Furthermore, FP-ABC can reduce VM migrations by nearly 6.1% when compared with ABC_RS. An unreasonable VM allocation policy will increase the frequency of VM migrations. Therefore, FP-ABC policy can find best hosts as BestTargetHostList in Δt. As a result, we can save more energy.

4.5. Evaluating the Energy-Efficiency of FP-ABC with Different VM Allocation Policies. In formula (9), ESV is an energy-efficiency comprehensive index. It measures the tradeoff between performance and energy consumption. As we all know, the QoS will be violated when energy is saved greatly and vice versa. Therefore, achieving a good energy-efficiency tradeoff is difficult. However, FP-ABC has a good performance in the metric of ESV depicted in Figure 5. The first reason is that FP-ABC policy can reduce the VM migration frequency to keep live workload balance in data centers; the second reason is that the proposed energy consumption evaluation model can be used to make a better evaluation of energy consumption in data centers; the third reason is that idea of simulated annealing and gradient descent can improve the performance of ABC based policy. Evaluation model [8] is presented as follows:

$$\text{ESV} = \text{EC} * \text{SLAV}. \tag{9}$$

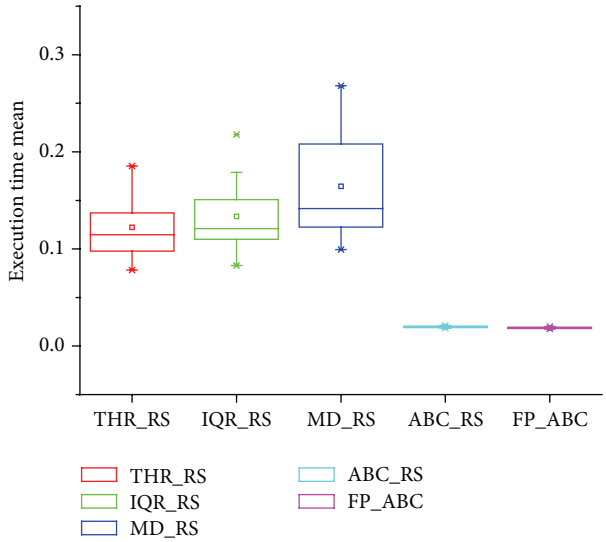


Figure 6: ETM metric of the five algorithms.

4.6. Evaluating the Decision Time of FP-ABC with Different VM Allocation Policies. In Figure 6, based on the index of execution time mean in Cloudsim3.0, it can reduce the decision time 85% to THR_RS, 86% to IQR_RS, 88% to MD_RS, and 13.8% to ABC_RS. Compared with these traditional migration policies, the FP-ABC is more effective to minimize the decision-making time.

5. Discussion

5.1. The Importance of Accelerating the Process of Finding Local Optimized Target Hosts with FP-ABC. A huge number of live migrant VMs are waiting to be allocated to their best target hosts in Δt. Therefore, the solutions should be given in some reasonable time window. Current state-of-the-art ABC based live VM allocation policy should be improved by accelerating

the speed of finding the local optimized target hosts. To solve this problem, the idea of gradient descent is adopted to ABC based live VM allocation policy. Current ABC based policy only makes the judgment based on its fitness value, without considering the requirement of speed. The idea of achieving a local solution quickly is realized by gradient descent. It will accelerate the process of searching local optimization in order to accelerate the speed of FP-ABC policy. In Section 4.6, we see that value of execution time mean is reduced when compared with 3 traditional policies and ABC_RS policy. It is significant when facing with amount of applications and also can save energy.

5.2. The Importance of Adopting Idea of Simulated Annealing to Find Global Optimized Target Hosts. We want to find the global optimum target host for each migrant VM in VM allocation, because the unreasonable target host will increase irrational energy consumption. In CloudSim3.0, three traditional VM migration policies are not good at finding the global optimum results, since their decision is made only by workload balance without reasonability analysis. Furthermore, the state-of-the-art ABC based policy is easy to fall into local optimum. Therefore, we add the idea of simulated annealing into ABC based policy, which can make scout bees jump out of the current solution space on a certain probability to detect other solution space.

5.3. Tradeoff between Energy Consumption and Service Quality. In data centers, to reduce energy consumption or improve the quality of service simply is not a good energy-saving policy, because reducing the total energy consumption will decrease service quality; that is to say, it will increase the value of SLAV. The main objective of this paper is to achieve energy efficient goal and make the value of SLAV accepted. Experimental results show that FP-ABC can save energy effectively in the balance of the EC and SLAV, which is based on formula (9) introduced in Section 4.5.

6. Conclusions

This paper presents a new live VM allocation policy to reach as far as possible to meet the SLAV value while saving energy consumption. We proposed the state-of-the-art ABC based policy with the idea of simulated annealing and the idea of gradient descent. The idea of gradient descent is used to find the optimal solution with higher speed. And the idea of simulated annealing is used to find the global optimal solutions in Δt. Simulation experiment has proved that FP-ABC policy can effectively reduce energy consumption with relative high level of QoS. In addition, FP-ABC can effectively reduce the energy consumption of data centers by 5.1%, when compared with the state-of-the-art algorithm of ABC_RS.

In the future, the proposed the state-of-the-art ABC based policy should be adapted to the cloud of cloud. Cloud of cloud is a new business model to get an economic data center with good service performance. The improved ABC based optimized model should be implemented in new environment.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The authors are grateful to the financial support by the National Natural Science Foundation of China (nos. 61202306, 61472049, 61402193, and 61572225), the Foundation of Jilin University of Finance and Economics (nos. XJ2012007 and 201401), the Education Department of Jilin Province "Twelfth Five-Year" science and technology research projects under Grant no. 2015410, and Social Science Foundation of Jilin Province under Grant no. 2015BS48.

References

[1] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proceedings of the 8th International Symposium on High-Performance Computer Architecture (HPCA '02)*, pp. 29–40, Cambridge, Mass, USA, February 2002.

[2] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp. 164–177, 2003.

[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[4] J. Zhao, L. Hu, Y. Ding, G. Xu, and M. Hu, "A heuristic placement selection of live virtual machine migration for energy-saving in cloud computing environment," *PLoS ONE*, vol. 9, no. 9, Article ID e108275, 2014.

[5] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Tech. Rep. TR06, Erciyes University, 2005.

[6] J. Jiang, Y. Liu, L. Wang, J. Chen, N. Huang, and X. Wei, "ABCC: an energy-efficiency VM consolidation algorithm based on heuristic backward artificial bee colony method in data centers," *Journal of Jilin University (Science Edition)*, vol. 6, pp. 1239–1248, 2014.

[7] X. Fan, W. D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 13–23, 2007.

[8] A. Boglazov, *Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing*, The University of Melbourne, Victoria, Australia, 2013.

[9] W. Deng, F.-M. Liu, H. Jin, and D. Li, "Leveraging renewable energy in cloud computing datacenters: state of the art and future research," *Chinese Journal of Computers*, vol. 36, no. 3, pp. 582–598, 2013.

[10] C. Fiandrino, D. Kliazovich, P. Bouvry, and A. Zomaya, "Performance and energy efficiency metrics for communication systems of cloud computing data centers," *IEEE Transactions on Cloud Computing*, 2015.

[11] H. Nakada and T. Hirofuchi, "Eliminating datacenter idle power with dynamin and intelligent VM relocation," in *Distributed Computing and Artificial Intelligence*, vol. 79, pp. 645–648, Springer, Berlin, Germany, 2010.

[12] K. Nishant, P. Sharma, V. Krishna et al., "Load balancing of nodes in cloud using ant colony optimization," in *Proceedings of the 14th International Conference on Modelling and Simulation (UKSim '12)*, pp. 3–8, IEEE, Cambridge, UK, March 2012.

[13] R. Jeyarani, N. Nagaveni, and R. Vasanth Ram, "Self adaptive particle swarm optimization for efficient virtual machine provisioning in cloud," *International Journal of Intelligent Information Technologies*, vol. 7, no. 2, pp. 25–44, 2011.

[14] R. Kapur, "A workload balanced approach for resource scheduling in cloud computing," in *Proceedings of the 8th International Conference on Contemporary Computing (IC3 '15)*, pp. 36–41, IEEE, Noida, India, August 2015.

[15] X. J. Bi and Y. J. Wang, "A modified artificial bee colony algorithm and its application," *Journal of Harbin Engineering University*, vol. 33, no. 1, pp. 117–123, 2012.

[16] PlanetLab, http://planet-lab.org/.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 9315672, 6 pages
http://dx.doi.org/10.1155/2016/9315672

Research Article

A User-Customized Virtual Network Platform for NaaS Cloud

Lei Xiao,[1] Yu Sheng,[1] Guanlan Tan,[1] Jianxin Wang,[1] and Yi Pan[2]

[1]*School of Information Science and Engineering, Central South University, Changsha 401083, China*
[2]*Department of Computer Science, Georgia State University, Atlanta, GA 30302-4110, USA*

Correspondence should be addressed to Yu Sheng; shengyu@csu.edu.cn

Received 25 February 2016; Accepted 18 May 2016

Academic Editor: Ligang He

Now all kinds of public cloud providers take computing and storage resources as the user's main demand, making it difficult for users to deploy complex network in the public cloud. This paper proposes a virtual cloud platform with network as the core demand of the user, which can provide the user with the capacity of free network architecture as well as all kinds of virtual resources. The network is isolated by port groups of the virtual distributed switch and the data forwarding and access control between different network segments are implemented by virtual machines loading a soft-routing system. This paper also studies the management interface of network architecture and the uniform way to connect the remote desktop of virtual resources on the web, hoping to provide some new ideas for the Network as a Service model.

1. Introduction

With the recent rapid development of cloud computing, more and more personal and enterprise users enjoy the various and convenient services brought by cloud computing. The demand based access model not only saves a lot of expenses for the user, but also avoids the waste of a large number of idle resources for the society as a whole [1]. Being necessary for the development of the cloud computing, virtualization technology abstracts physical resources into logical resources, makes the diversity and compatibility of equipment transparent to the upper structure, and realizes more granular utilization of resources, including calculation, memory, disk, network, and operating system.

Along with the advance of virtualization technology, cloud providers are able to meet more requirements of computing and storage capacity. Meanwhile, great efforts have also been made to do the research in this respect [2]. Data networks, considered a necessary component of the infrastructure that links the consumable resources together, have not been utilized sufficiently compared to other resources. While the computing and storage resources have their flexible consumption model [3] in most IaaS (Infrastructure as a Service), the network resource has not yet been turned into a consumable service from the user perspective.

It is necessary to start with the reference to the concept of network virtualization, which is the focus of the paper [4]. The network virtualization generally refers to the technology used to abstract the physical network resources. It makes the physical network resources pooled, which allocates resources to users according to their requests in a logical way, which can achieve the goal of flexible resources division or merger. The users can manage their allocated virtual networks independently without the concerns about the physical implementation detail and the network resource conflict with other users. The technologies commonly used in the implementation of network virtualization are VLAN (Virtual Local Area Network), VPN (Virtual Private Network), Overlay Network, programmable network, and the currently hot SDN (software-defined networking) [5] and DCN (data center network) [6, 7]. Unlike the above, our research objective is to connect the virtual network to the virtual machine inside the cloud.

The cloud computing service model has three basic services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Similarly, in [8], the authors proposed a new concept on the network, Network as a Service (NaaS). According to the authors, NaaS should allow a user to request a network by specifying precisely

the network topology, the router computing capacities, the routing protocols, and possible other features.

Costa et al. [9] think that users now have little control over the network, so they suggest that NaaS should be network visibility and custom forwarding. As for the network visibility, users are provided with an abstract view of their allocated VMs without the need for reverse-engineering, but with easier deployment. As for custom forwarding, users can implement custom routing protocols, design firewall [10], and develop gateway, such as acceleration gateway [11].

In this paper, based on the above concepts, we present our design and implementation of VNetCloud, a platform that allows users to set up their own network infrastructure in a manageable way and makes the network a consumable service for users. VNetCloud is a virtual cloud platform with the network as the core requirement. The user platform will get an exclusive, isolated cloud environment. In VNetCloud, users can organize their computing, storage, and network resources, especially their network resources in their exclusive space. VNetCloud is not a local area network (LAN) but a collection of networks.

The contribution of our work can be summarized as follows:

(1) We designed and implemented a NaaS oriented platform prototype, which provides users with not only an isolated virtual resource environment, but also a flexible network topology, which enables a user to freely combine and dynamically dispatch his virtual resources according to the requirement.

(2) We have used the HTML5 based drawing technology to provide users with a more visual and convenient interaction environment, in which the users can drag and drop to add or manage virtual resources and topological relations between virtual resources, thus bringing about the implementation of high autonomy in operations management.

(3) We have studied and implemented a remote desktop method based on HTML5, which is used to push all kinds of the system desktop environments to a unified web interface through the protocol conversion middleware and therefore provides a strong support for the platform.

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 illustrates the architecture and implementation of VNetCloud. In Section 4, we describe application scenarios of our platform. We conclude the paper in Section 5.

2. Related Work

VPC (virtual private cloud), an on-demand configurable pool of shared computing resources allocated within a public cloud environment, was first put forward as a commercial solution by AWS [12]. It can divide for each user on public cloud an exclusive VLAN, and the user can add cloud services to this private network space, create a subnet, and establish connection with enterprise internal private cloud through VPN. Based on the idea of VPC, Wood et al. proposed the idea of CloudNet architecture [13], which joins VPNs with cloud computing, with VPNs providing secure communication channels and allowing customer's control over network provisioning and configuration. CloudNet can provide secure and seamless cloud resources to enterprises.

In [8], the authors proposed an extension of the virtual private cloud concept, that is, Cloud Networking. In this paper, the authors generalize the ideas in [13] and allow cloud and network resources to be handled as a single set. Network here refers to the connection between clouds, which are the nodes in the net. NCSS (Network-Aware Cloud System Suite) proposed by the authors has an integral control over clouds and the network between clouds, providing functionalities such as distributed network and cloud resource discovery, network and cloud mapping and creation, network and computing monitoring, and network and cloud resource management. The authors' research is based on the proposal that the whole routing topology information and physical routing equipment (such as PE and CE) information be obtained or set up on the same control platform.

CloudNaaS is a network service platform that allows users to utilize many network functions [14]. The authors present a design of an integrated provisioning system for cloud applications and network services with interfaces for customers to specify network requirements. CloudNaaS leverages the OpenNebula [15] cloud framework to implement the cloud controller component and utilize OpenFlow [16] enabled switches within lab-base setup. CloudNaaS primitives are implemented within the cloud infrastructure itself using high-speed programmable network elements.

In [17], the authors put forward some challenges and applicable scenes concerning Network as a Service and design and implement a cloud platform with NaaS through exploiting the functionality provided by software-defined networks, hoping that the cloud provider can integrate the cloud controller into an integrated network manager module to have a centralized view of the network, which implements the NaaS paradigm and is responsible for serving all the network related requests including queries.

In recent years, many researches are about how to enable the user to have control over different data centers or all kinds of transmission equipment between cloud networks or have implemented the flexible control of Network traffic based on software-defined network, making the network more intelligent. But it is difficult to make common users be able to fully control all kinds of transmission equipment now. To most of the requirements of the enterprises, all their resources are not distributed on different clouds. The implementation of their requirements for access control, security isolation, load balancing, and so forth does not necessarily rely on the control of transmission equipment like PE route, for soft-routing can be provided to users as a virtual machine in the public network. In this paper, what we are going to solve is aiming at the solutions to the free organization of the logical relations between their virtual resources, even the construction of complex network architecture, and the deployment of all kinds of security strategies for large users on public clouds.

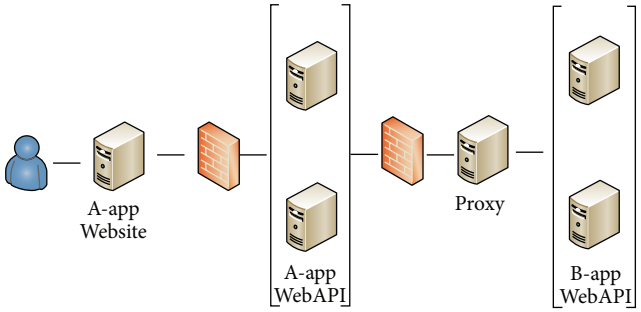


Figure 1: A network topology of a commercial system.

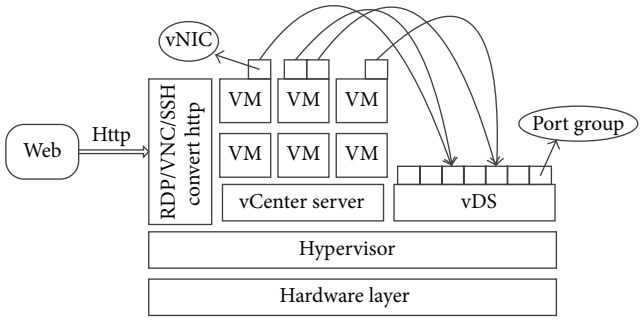


Figure 2: The infrastructure of VNetCloud.

3. The Architecture and Its Implementation

In our platform, the user can not only use the virtual machines resource in cloud, but also build their network topology for their business. As shown in Figure 1, it is a network topology of a commercial system.

Let us call this mobile application A. When it is in use, it first visits the website server on the outer network. The authentication and authorization servers, along with the WebAPI server of A-app, are set up in a security interception area surrounded by two layers of firewall. Passing through the security zone, mobile application A also needs to get access to the load balance module of B-app and the gateway before getting access to basic services, which is the production environment architecture of the product, and the test environment of the product must have same property. Here is just one product of the company. When the production environment is a complicated environment with a high cost, the test environment needs an equivalent investment.

As we know, test environment is different from the production environment, which is a transient operating environment, instead of being used in software iterative cycle. This company may require a test environment charged by demand and time. While the mainstream cloud service providers provide users with services focusing on computing and storage, to build such a complex test environment on a public cloud platform will be something very painful and difficult.

In this paper, we aim to provide users with a virtual cloud platform which is made by virtual machines and virtual network devices. With the platform, users can build a test environment for the commercial system.

The types of customers the public cloud platform faces are various, and their demands for network are varied. Considering that the existing cloud providers have quantitative billing on resources such as computing, memory, and hard disk, it is natural to list the network as one of the quantifiable resources.

3.1. The Platform Infrastructure. As a prototype, we choose VMware vSphere as our virtualization solution. The vSphere, as a mature commercial product, not only features the most advanced and complete server virtualization technology, but also offers the API and the SDK for these products and components. This platform customization management of vSphere can be implemented with the use of VMware vSphere Web Services SDK [18], which provides management interfaces for ESXi and vCenter and contains a Java sample code and very detailed API documentation. As shown in Figure 2, it is the architecture diagram for VNetCloud.

In addition, the VDS (vSphere distributed switch) [19] is deployed for the management of virtual network, and the collection of network segments is differentiated through the VDS port group. The distributed port groups group many ports under a common zone and provide locating point for a virtual machine connected to the marked network [20]. The virtual machine does not connect its virtual network interface card (vNIC) to a specific port on the vSwitch but to a port group. All the virtual machines connected to the same port group belong to the same network in the virtual environment. The port group can be used to break up the broadcast domains in the virtual environment. The vNIC of different port groups will not be broadcast by each other. As a result, the package transmission in the same port group is more secure.

Apparently, this virtual switch system is based on the Forwarding and Control Element Separation (ForCES) [21] framework, which can be divided into two logical parts, data plane and control plane. Data plane performs the actual operations such as packet switching, filtering, and marking. Control plane is a central controller of virtual network management, allowing administrators to manage network services through the higher-level functionality.

3.2. Web Interface. Our platform highlights that it is free for users to organize the isolation network relations between the virtual resources they have. Therefore we not only give the infrastructure support, but also make users feel intuitively that this network is a user-customized platform.

KineticJS is an open source JavaScript framework based on HTML5 [22, 23]. It encapsulates HTML5 Canvas [24], allowing easy and convenient operations of HTML5 Canvas. In KineticJS, users can draw, operate, and modify all graphics or pictures on the canvas and add event monitors on them.

We design an HTML5 based drawing panel as the main interface that allows users to deploy their customized network topology. We can drag and drop the preset virtual device icon from one division to the center panel through KineticJS. The user, through a drag-and-drop operation, can freely combine his resources and connect a virtual device

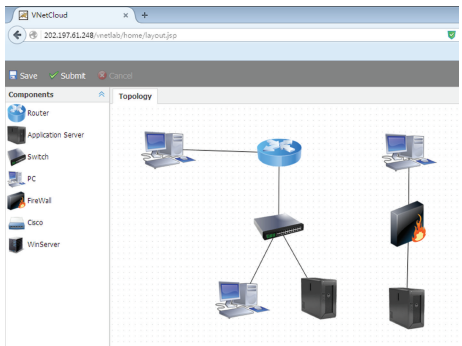


Figure 3: Web interface of the platform.

icon to another, which means that the two devices can be put in the same port group. By clicking on the linked line icon, the corresponding network interface relation on both ends of the device can be set. And by clicking on the device, the configuration of virtual resources such as CPU, memory, hard disks, and network interfaces can be viewed and set up. We also place the entrance to the remote desktop in them. Figure 3 shows the result of the implementation of the platform interface in a user-customized way.

The virtual devices on the canvas and their dragging, drawing, and connection are all based on KineticJS framework, implementing instances such as Kinetic.Topology, Kinetic.Topology.Background, Kinetic.Topology.Device, and Kinetic.Topology.Line. Kinetic.Topology is the main class of the topological graphs, containing the device list and the line list and the connectors between them and the get and set methods of these objects. When a device is dragged into the canvas, the set method of the device object is invoked, and the canvas will be updated. There are two layers in Kinetic.Topology, used to display, respectively, graphics painted and message data. Kinetic.Topology.Device is a device class, and it also has a line list, different from the line list in Kinetic.Topology, which is the relevant line to the device. In addition, it also contains removing, checking range, binding event, drawing, and so on. Such a visual usage scenario can strengthen the user's global mastery of the virtual environment.

3.3. Logic of the Customized Network Topology. After the topology design in the web interface is finished, the data of connecting relation can be sent to background. With toJSON method of Kinetic Topology, we can get the JSON data of our customized network topology, which will be a collection of Device and Line. Each subset in Device includes all kinds of configuration information of the virtual devices and their coordinates on the canvas. Each subset in Line contains the network interface information of virtual machines on both ends of all lines; those JSON data are submitted to the background, through the Java reflection mechanism and the deserialization of JSON, and a collection of Device and Line is aggregated to become the object of network topology.

In the customized network topology, normally we put the network interface of the virtual devices attached to the same line into the same port group. Since a device can have multiple virtual networks, the different network interfaces can be placed in different virtual networks, and network interfaces of virtual machines in the same port group can communicate with each other in the same subnet. Through VDS, we can connect the network interfaces, which are necessarily to be put in the same LAN, to the same port group at the same time.

As for network isolation and access control, we need a soft-routing system as a customizable gateway device, which can be loaded to a virtual machine. So we choose openWRT [25], to be the core device of our network partitioning that can completely meet the complex requirements of network architecture for users. By joining different network interfaces of openWRT to different port groups, the network data between the port groups can be forwarded by openWRT. Like using other common Linux systems, using openWRT allows more flexible and diversified connections of the configuration between different subnets for the user.

3.4. Cloud Desktop. VNetCloud, as a one-stop virtual cloud platform, is expected not only to implement the overall control over the network architecture of the resources the users have, but also to get easy access to virtual resources. A click of the device icon on web interface will implement the access to the virtual machines of different systems on the browser [26, 27].

Guacamole is an open source web application based on HTML5 [28]. Guacamole pushes various types of desktop environment by proxy to the unified web page through the protocol conversion middleware. It is fused with our platform well. The user can use VNetCloud in any place where he can get access to the Internet with a computer or mobile phone via a browser, without the need to install any client and plug-in.

Guaca mainly consists of three parts: Guacamole protocol, guacad, and web application. Guacamole protocol contains remote display rendering and event transmission. Guacd is a daemon process responsible for protocol conversion, the core of Guacamole, which dynamically loads various remote desktop protocols. Web application is presented at the front. Through the integration of Guaca on the platform, we simply click on the remote desktop interface of a device on the topology with the browser, and we can view the remote desktop in a new window or embedded in iframe and have access to it.

When a user has dozens of virtual devices, it may be difficult for him to locate the machine he wants to visit, so it costs time to check back. The combination of network topology and cloud desktop makes users' access target clearer.

4. Application

An important application of the NaaS cloud is the university network experiment. We first build an experimental environment for the computer network course in our university by the VNetCloud. Computer network, as one of the core courses of the Information Specialty, is abstract and difficult to understand. At many times, it requires not only theoretical teaching but also lab practicing. On one hand, the computer network lab currently is taught in laboratory and requires

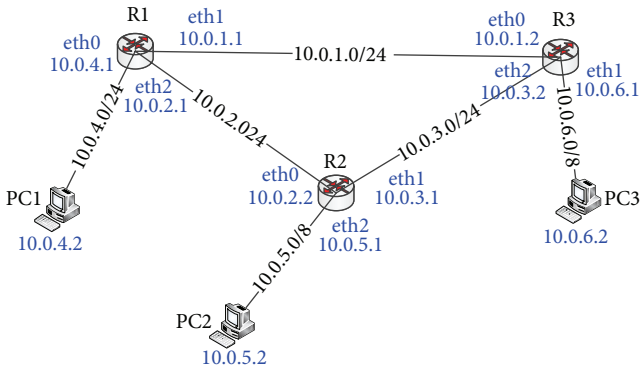


Figure 4: A topology of RIP experiment.

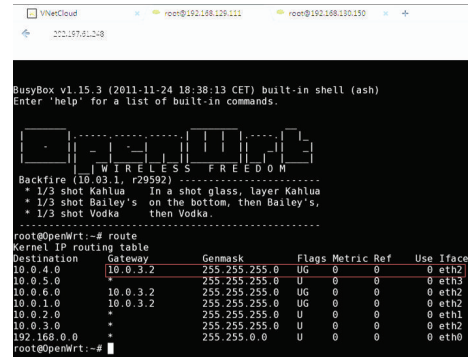


Figure 5: R2 routing table.

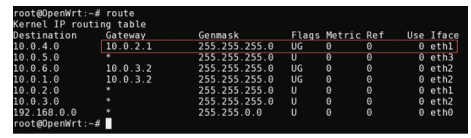


Figure 6: The change of R2 routing table after we start up the eth2 of R1.

a lot of resources. On the other hand, the simulation based experiments cannot show the real data package transmitting in the physical networks [29]. VNetCloud provides students with a customized virtual network environment for computer network experiments, where students can design a certain network topology, deploy a new routing protocol, and reconfigure the routing rules. Then VNetCloud generates the corresponding real network by connecting virtual machines. From the generated network, students can learn the flows of data packets and analyze the working process and performance of protocols by using packets capturing tool.

In this section, the Routing Information Protocol (RIP) experiment is used as an example to illustrate the usage of VNetCloud. In Figure 4, the network topology contains three routers and three PCs. The design of the network topology as shown in Figure 4 can be simplified, done by dragging the device icons from the web interface of VNetCloud and connecting them. Then, we can click the "Save" button. The background of VNetCloud will assign the demanded resources (three routers and three PCs) for the user. In this scenario, the resources will be released if the students exit. The user can specify a pair of network cards which are responsible for the connection between two devices, by clicking on the "Connection" icon. For example, in Figure 4, eth1 and eth3 are the pair of network cards responsible for the linking between PC1 and PC2.

When the network environment is ready, RIP experiment can be carried out. In the initial state, PC2 (10.0.5.2) is unable to access PC1 (10.0.4.2). By clicking the icon of router R2 and the access button of the pop-up device configuration window, we can remotely access R2 through cloud desktop. Through the access of R2, we can learn the routing table of R2. The routing table displays the network information that can only be directly accessed by R2, without knowing the network segments 10.0.1.0, 10.0.4.0, and 10.0.6.0. Thus, the students can install and configure Zebra and RIP for three routers. To learn the response of RIP to the network topology change, we can shut down eth2 of R1 and open the quagga service of three routers. Then, we can observe that the routing table of R2 will change soon. As shown in Figure 5, three pieces of new routing information are added in the routing table of R2 with "R" flag. The routing information with "R" flag denotes that it is produced by the RIP dynamic routing. Due to the shutdown of eth1, R2 cannot access R1 directly. It can access the network segment 10.0.4.0/24 through the route from R3.

In the next operation, we restart eth2 of R1. It is observed that the routing table of R2 changes accordingly. As we can learn from Figure 6, R2 can access R1 directly. It can access the network segment 10.0.4.0/24 through routing to R1.

Besides the computing network experiment, the platform can also be used to simulate a commercial system, even to deploy a complex commercial system.

5. Conclusion

In this paper, we design a user-customizable virtual network platform that enables users to simulate real network environment to build topology for project testing according to the production environment, to undertake penetration testing and network protocol testing or design a more secure cloud for one's own application. In VNetCloud, physical infrastructure is built by the system, and the user only needs to focus on the network architecture, which is very convenient for deploying a complex network architecture on the public cloud.

To simplify the platform interactivity, we provide users with a more intuitive and convenient operating environment in which many operations can be easily done by dragging and dropping the icons, so that the user network resources, along with their dynamic relationships and operations, can be clearly displayed. In addition, concerning the method used to connect to the virtual resources, an open source project Guacamole integrated into this platform. Thus the users can access the remote desktop of the corresponding device in the customized topology with a web browser.

In the future, we will also add such functionalities to the platform as user-customizable device icons, uploading image files, and deployment of OVF (Open Virtualization

Format) template, allowing the user to independently add virtual devices to the customized web space, making the user really feel that this is his unique, personalized cloud when enjoying public cloud services.

Competing Interests

The authors declare that there is no conflict of interests.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grant nos. 61202494, 61402541, 61402542, and 61572530.

References

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proceedings of the Grid Computing Environments Workshop (GCE '08)*, pp. 1–10, November 2008.

[2] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, October 2011.

[3] S. S. Manvi and G. K. Shyam, "Resource management for Infrastructure as a Service (IaaS) in cloud computing: a survey," *Journal of Network & Computer Applications*, vol. 41, no. 1, pp. 424–440, 2014.

[4] M. F. Bari, R. Boutaba, R. Esteves et al., "Data center network virtualization: a survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[5] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an SDN platform for cloud network services," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 120–127, 2013.

[6] T. Zhang, J. Wang, J. Huang, Y. Huang, J. Chen, and Y. Pan, "Adaptive-acceleration data center TCP," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1522–1533, 2015.

[7] T. Zhang, J. Wang, J. Huang, Y. Huang, J. Chen, and Y. Pan, "Adaptive marking threshold method for delay-sensitive TCP in data center network," *Journal of Network and Computer Applications*, vol. 61, pp. 222–234, 2016.

[8] J. C. Soares, M. Melo, R. Monteiro, and S. Sargento, "Building virtual private clouds with network-aware cloud," in *Proceedings of the 5th International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP '11)*, Lisbon, Portugal, November 2011.

[9] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, *NaaS: Network-as-a-Service in the Cloud*, USENIX, 2012.

[10] Y. Cheng, W. Wang, G. Min, and J. Wang, "A new approach to designing firewall based on multidimensional matrix," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 12, pp. 3075–3088, 2015.

[11] P. Dong, J. Wang, J. Huang, and H. Wang, "Split-TCP based acceleration gateway over packet lossy networks," *China Communications*, vol. 12, no. 5, Article ID 7112033, pp. 100–112, 2015.

[12] L. M. Surhone, M. T. Tennoe, and S. F. Henssonow, *Amazon Virtual Private Cloud*, Betascript Publishing, 2010.

[13] T. Wood, A. Gerber, K. K. Ramakrishnan, P. Shenoy, and J. V. D. Merwe, "The case for enterprise-ready virtual private clouds," in *Proceedings of the Conference on Hot Topics in Cloud Computing (HotCloud '09)*, article 4, USENIX Association, 2009.

[14] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "CloudNaaS: a cloud networking platform for enterprise applications," in *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, article 8, Cascais, Portugal, October 2011.

[15] D. Milojičić, I. M. Llorente, and R. S. Montero, "OpenNebula: a cloud management tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 11–14, 2011.

[16] E. Watanabe, K. Ando, I. Karube, H. Matsuoka, and S. Suzuki, "Network innovation using openflow: a survey," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, no. 1, pp. 493–512, 2014.

[17] D. Kakadia and V. Varma, "Network virtualization platform for hybrid cloud," in *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom '13)*, vol. 2, pp. 69–74, December 2013.

[18] VMware vSphere Web Services SDK, https://www.vmware.com/support/developer/vc-sdk/.

[19] VMware vSphere Distributed Switch (VDS), http://www.vmware.com/products/vsphere/features/distributed-switch.

[20] S. Zhou, "Virtual networking," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 4, pp. 80–85, 2010.

[21] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework," 2004, https://www.ietf.org/rfc/rfc3746.txt.

[22] Z. Ma, C.-Y. Yeh, H. He, and H. Chen, "A web based UML modeling tool with touch screens," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*, pp. 835–838, Västerås, Sweden, September 2014.

[23] KineticJS, http://www.kineticjs.com/.

[24] M. Grady, "Functional programming using JavaScript and the HTML5 canvas element," *Journal of Computing Sciences in Colleges*, vol. 26, pp. 97–105, 2010.

[25] M. Chmielewski, P. Szyszkowski, and M. Piechowiak, "Application of IP multicast in embedded systems with openWRT," in *Proceedings of the 8th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP '12)*, pp. 1–5, Poznan, Poland, July 2012.

[26] L. Deboosere, B. Vankeirsbilck, P. Simoens, F. De Turck, B. Dhoedt, and P. Demeester, "Cloud-based desktop services for thin clients," *IEEE Internet Computing*, vol. 16, no. 6, pp. 60–67, 2012.

[27] I. Alimirza, A. K. Tripathy, V. Sarang, A. Joshi, and S. Shah, "ProtoCloud: a cloud based desktop," *Computer Science & Information Technology*, vol. 1, no. 1, pp. 57–64, 2013.

[28] D. Mulfari, A. Celesti, M. Villari, and A. Puliafito, "Providing assistive technology applications as a service through cloud computing," *Assistive Technology*, vol. 27, no. 1, pp. 44–51, 2015.

[29] N. Yalcin, Y. Altun, and U. Kose, "Educational material development model for teaching computer network and system management," *Computer Applications in Engineering Education*, vol. 23, no. 4, pp. 621–629, 2015.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 4670271, 12 pages
http://dx.doi.org/10.1155/2016/4670271

Research Article

FPGA-Aware Scheduling Strategies at Hypervisor Level in Cloud Environments

Julio Proaño Orellana,[1] Blanca Caminero,[1] Carmen Carrión,[1] Luis Tomas,[2] Selome Kostentinos Tesfatsion,[2] and Johan Tordsson[2]

[1]*Computing Systems Department, University of Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete, Spain*
[2]*Department of Computing Science, Umeå University, 901 87 Umeå, Sweden*

Correspondence should be addressed to Julio Proaño Orellana; julio.proano@alu.uclm.es

Received 25 March 2016; Revised 13 May 2016; Accepted 22 May 2016

Academic Editor: Florin Pop

Current open issues regarding cloud computing include the support for nontrivial Quality of Service-related Service Level Objectives (SLOs) and reducing the energy footprint of data centers. One strategy that can contribute to both is the integration of accelerators as specialized resources within the cloud system. In particular, Field Programmable Gate Arrays (FPGAs) exhibit an excellent performance/energy consumption ratio that can be harnessed to achieve these goals. In this paper, a multilevel cloud scheduling framework is described, and several FPGA-aware node level scheduling strategies (applied at the hypervisor level) are explored and analyzed. These strategies are based on the use of a multiobjective metric aimed at providing Quality of Service (QoS) support. Results show how the proposed FPGA-aware scheduling policies increment the number of users requests serviced with their SLOs fulfilled while energy consumption is minimized. In particular, evaluation results of a use case based on a multimedia application show that the proposal can save more than 20% of the total energy compared with other baseline algorithms while a higher percentage of Service Level Agreement (SLA) is fulfilled.

1. Introduction

Nowadays, the processing computational resources have shown an important change to achieve a balance between performance and power consumption. Devices such as GPUs and FPGAs among others are integrated as specialized processing elements into cloud environments to extend the capacity of the cloud.

FPGAs are commercial off-the-shelf reconfigurable silicon devices that achieve hardware-like performance with software-like flexibility. These are facts of great interest in the context of cloud computing. Cloud mainly leverages virtualization technology (i.e., virtual machines) to manage resources of a data center. Thus, providers can share the physical resources between clients.

In this context, resource scheduling is a critical issue that can contribute to increasing the benefits of cloud platforms. On one hand, depending on how many resources are allocated to different users' requests for service, more or fewer requests can be serviced while fulfilling their SLA. This fact impacts the benefits obtained by the cloud provider. On the other hand, additional advantages can exist, such as achieving a positive effect on the data center energy consumption. The combination of both implies getting a better Return on Investment (ROI) from the infrastructure which is crucial when serving Software as a Service (SaaS) requests with QoS requirements. Typically, a SaaS user issues a request for a specific service, with particular QoS-related constraints (i.e., a deadline). The user does not need to be aware of which type and how many resources are required to get its response. The system must be able to allocate and schedule the necessary resources (both in quantity and type) to serve this request.

This paper addresses the scheduling of jobs within a cloud infrastructure which is composed of heterogeneous physical nodes (with and without FPGA accelerators) from a hierarchical point of view. In this structure the two levels of scheduling are

(1) the Cluster Level Scheduler (CLS), to decide the type of node where the virtual machine (VM) that will serve the request will be deployed,

(2) the Node Level Scheduler (NLS), to decide which one of the VMs allocated to the node will get the use of the accelerators (FPGAs in particular) available in a node.

The problem of high-level scheduling among physical nodes, referred to as CLS, has previously been addressed by the authors in [1, 2]. Now, this work focuses on the impact of different scheduling strategies within a physical node with FPGA resources, which are applied at the hypervisor level. In a nutshell, the main contributions of the paper are the following:

(i) to extend the hypervisor functionality to support dynamic control of FPGA devices as cloud computational resources,

(ii) to propose a novel FPGA-aware Node Level Scheduler (NLS) metric aimed at QoS provision while reducing energy consumption,

(iii) to present a novel fine-grained FPGA-aware Node Level Scheduler,

(iv) to evaluate the Node Level Scheduler proposals in a real testbed, comparing them to some simple scheduler techniques.

This paper is structured as follows. Sections 1 and 2 introduce and motivate the problem being tackled. Section 3 reviews literature related to the topic of the paper. The framework where the presented research has been carried out is outlined in Section 4. Next, Section 5 provides the details on how accelerators (and in particular, FPGAs) are included into the scheduling strategies, and the additional fine-grained level of scheduling is introduced in Section 6. The evaluation of results are presented in Section 7. Finally, Section 8 provides some concluding remarks.

2. Background and Motivation

During the last few years, cloud computing has consolidated as a paradigm that enables a flexible and on-demand use of IT resources at different levels (infrastructure, platform, or software) as a service. Typical cloud providers support their service by means of massive data centers (usually spread around the globe), while cloud users get access to the resources with a pay-per-use model. This is referred to as a public cloud model. The cloud computing paradigm can also be applied within an organization, leading to the private cloud deployment model. Resources are pooled and shared among the different organization user groups to meet their demands on IT resources. In either case (public or private), cloud computing platforms are composed of pools of computing, storage, and networking resources that are made available as a service to their users. Service Level Agreements (SLAs) are established between providers and users, to specify the conditions of the service, both from technical (availability, Quality of Service, ...) and formal (cost of the service, penalties in case of contract breach, ...) points of view.

Many challenges exist in this model that turn out to be the focus of many research projects, such as efficient resource management [3], security and privacy concerns [4], or standardization [5], just to name a few of them. In particular, some kind of orchestration is needed in order to allocate the adequate resources to every user request [6]. As it has been pointed out before, cloud providers face the following dilemma: admitting more users into the system would lead to more incomes, but if the available resources are not enough for fulfilling the established SLAs, penalizations will cut benefits down. Thus, care must be taken when admitting more users into the system.

From another point of view, data center's energy consumption is nowadays a big concern. Energy costs are a major contributor to a data center's Total Cost of Ownership (TCO) [7]. Thus, strategies to improve data center's efficiency are the topic of many research efforts, such as adjusting the voltage supplied to servers according to their workload [8] or even creating specific hardware designs [9]. One strategy that can provide benefits regarding energy consumption as well as performance in data centers is the integration of accelerators as specialized resources within the cloud infrastructure [10], such as General Purpose Graphics Processing Units (GPGPUs) or Field Programmable Gate Arrays (FPGAs).

FPGAs allow cloud computing data centers to scale up in a more efficient way than using just conventional processors [11]. It is interesting to note that the initiatives towards building exascale systems with low-energy consumption driven by the European Union (EU) are based on the integration of FPGAs with ARM processors [12].

However, using heterogeneous resources in cloud environments is still an open challenge, as illustrated by the work presented by Microsoft in [13]. Frequently, cloud applications which are running on VMs with a fixed amount of hardware resources including accelerators do not use the same number of them all the time due to their features. Thus, a proper scheduling strategy might help to optimize the utilization of these resources.

Moreover, different applications could benefit from the FPGA over time, depending on their degree of compliance with QoS requirements. To achieve this objective, the FPGA should be properly shared among competing service requests.

3. Related Work

Incorporation of FPGAs in the cloud context is a relatively new area of research. So, the Catapult project [13] led by Microsoft represents the first detailed investigation of applying FPGAs within an enterprise-level data center application. In this case, FPGA-accelerated nodes are used for the Bing web search engine. Results show great improvements in both the latency and the throughput of the service.

Nevertheless, FPGA-aware scheduling in a cloud environment is still at initial stages. Integrating FPGAs as first class computational resources to provide cloud service demands novel high-level programming models to simplify the development of software applications, maximizing communication to FPGAs, and the development of efficient FPGA-aware scheduling algorithms.

One crucial step to efficiently run applications across heterogeneous hardware is to provide optimized versions of computational kernels (as BLAS routines or FFT) [14]. OpenCL [15], VIVADO [16], and Lime [17] are high-level programming models focused on reducing the time-to-market in the designing process.

In addition, providing FPGAs as cloud computing resources demands virtualization support in hardware with

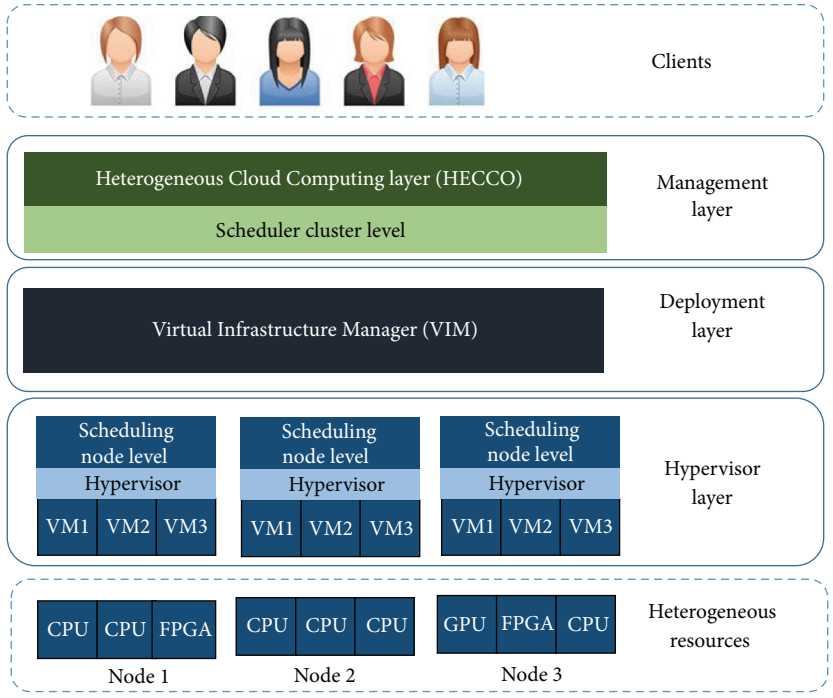


Figure 1: Architecture overview.

near-native performance. Open source interface frameworks have recently emerged (RIFFA [18], DyRACT [19]) that enable FPGA designs to be accessed through an abstracted software API on the host with communication throughput between the host and FPGA close to the limits of modern PCIe interfaces.

Moreover, efficient resource managements and scheduling algorithms are open key challenges to providing FPGA cloud services. The scheduling of jobs faces a multiobjective optimization problem in the process of assigning jobs to a pool of limited resources. Resource management and scheduling have been a hot research topic in a cloud context, and many approaches have been proposed to place the VMs to physical CPU hosts. A survey of the most recent proposals about metaheuristic scheduling solutions for cloud can be found in [20].

In this context, cloud-centric integration of FPGAs frameworks is developed in [11, 21] by extending the open source cloud management system OpenStack [22]. The systems support the on-demand deployment of a user designed custom hardware while maintaining the cloud computing benefits of scalability and flexibility. The bitstream of the application is considered as a special VM image. So, FPGAs are eligible as computational resources by the clients.

Another recent related work presents a framework that integrates FPGAs in a standard server with virtualized resource management and communication [23]. The resource management selects the smallest reconfigurable FPGA area able to attend the request. If no one is found, the user request is rejected, and the request can be processed in software. As an application case study, they built a MapReduce accelerator for word counting and preliminary results are evaluated about integrating FPGA devices in the cloud using partial reconfiguration.

At some points, the systems mentioned above can be complementary to the work presented in this paper because all of them focusing on supporting FPGAs as computational cloud devices. But in contrast, in our case FPGAs are not visible to the clients. The idea is to provide a QoS Software as a Service by making efficient use of FPGA-aware scheduling algorithms. Our work focuses on developing a cloud management framework able to scale up and down FPGA devices in a dynamical way.

4. Hierarchical Scheduler Architecture Overview

In previous works [1, 2] the problem of integrating FPGAs into a cloud was faced by "HECCO" (Heterogeneous Cloud Computing) architecture. In those approaches, every node of the cloud system is composed of one or more CPUs with a certain number of cores each and zero or more accelerators. Frequently, the number of accelerators available within a node is lower than the number of CPU cores available. Therefore, they must be shared between clients. HECCO enables the provision of IaaS and SaaS services with Quality of Service (QoS) requirements.

In this work, the proposal architecture leveraged the use of accelerators to (a) finish job tasks within a particular time frame in order to fulfill their QoS requirements, expressed as Service Level Objectives (SLO) within a Service Level Agreement (SLA), and (b) reduce the energy footprint of cloud data centers.

The whole picture of the framework is depicted in Figure 1. It is organized into several layers:

(i) The *Management Layer* is responsible for receiving client requests and selecting the most suitable node

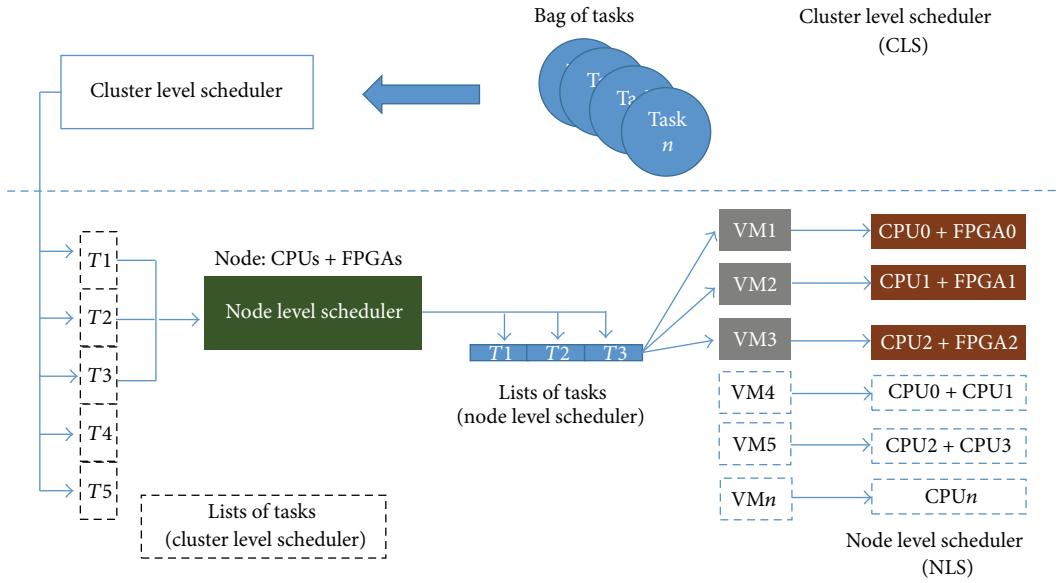


Figure 2: Different scheduling levels.

to allocate their applications. Each request is defined by a template in which parameters such as the type of service and the utilization time are defined. This layer is composed of the Heterogeneous Cloud Computing (HECCO), which includes the Cluster Level Scheduler (CLS).

As outlined above, it provides the "intelligence" to fulfill the QoS requirements expressed by clients within their SLAs. In particular, the CLS implements a series of mechanisms that are mainly aimed at allocating every request to the most appropriate resource (i.e., a node with or without accelerators). Its decision is based on the client's SLA and the availability of the resources.

More details on the management layer can be found in [2].

(ii) The *Deployment Layer*, which receives instructions from the management layer, is responsible for performing actions such as the creation and monitoring of the whole virtual environment. It has a complete view of the cloud and is composed of the Virtual Infrastructure Manager (VIM). The VIM is a centralized data center manager that allows to build and manage the cloud environment.

(iii) The *Hypervisor Layer* provides the control of the physical resources. This layer has a local view of the system and is composed of a hypervisor, which is responsible for the supervision of the VMs and resources for each node. This layer also contains a *Node Level Scheduler* (NLS) that is responsible for sharing the local resources within a node. In this case, the scheduling's decision is focused on the management of the use of the accelerators within each node, in order to get the most out of them.

More details about this element are provided in Section 5.

Figure 2 depicts the two different levels of scheduling. First, the CLS receives the tasks and selects the most suitable node to allocate each of them, following the strategies proposed in [2]. Second, the NLS manages the tasks allocated to a specific node.

More precisely, it selects the task which is going to receive the accelerator and for how long, by certain metrics.

In other words, given the fact that there may be more virtual machines allocated to a node than accelerators available, which virtual machine will deserve the use of an accelerator? In the rest of this work, some insight will be given on this issue.

5. Node Level Scheduling

In this section we will focus on the dynamic scheduling strategy for allocated jobs within a node and timing-related indicators, such as a deadline. This deadline is related to when the job must have finished in order to deem its SLA as fulfilled.

The main objective of the node level scheduling strategy is twofold: first, to meet application SLOs and second, to consume as less energy as possible. To achieve this multiobjective, the scheduling algorithm is aware of the properties of the physical nodes and due to the good performance and low-cost energy offered by the FPGAs, the scheduling maximizes their utilization.

Basically, the algorithm uses a heuristic process for assigning the physical resources inside the node (FPGAs and CPUs) to the virtual machines that will be deployed at each point of time. It is worth to mention that the use of this simple algorithm avoids the overhead of the system. Recall that the NLS is implemented within the hypervisor and should be able to run with minimal computational resources.

Require: Task_i, $i = 1, 2, \ldots, K$: Task T_i to be executed in the local node
$\text{List}_{\text{Task}}$ = Set of tasks
metric(Task_i): function which computes the metric for task Task_i
include(Task_i, $\text{List}_{\text{Task}}$): function which includes Task_i in $\text{List}_{\text{Task}}$
rank($\text{List}_{\text{Task}}$): function which sorts $\text{List}_{\text{Task}}$ according to the selected metric
top($\text{List}_{\text{Task}}$): function which extracts the first VM from $\text{List}_{\text{Task}}$

```
(1)  loop
(2)    Wait until FPGA_status == Free
(3)    for each Task_i/i ∈ 1 ··· N do
(4)      Task_i.metric = metric(Task_i)
(5)      include(Task_i, List_Task)
(6)      rank(List_Task)
(7)    end for
(8)    VM_first = top(List_Task)
(9)    attach FPGA to VM_first
(10) end loop
```

Algorithm 1: Node level scheduling algorithm.

In other words, a matching is done between computational resources and tasks (deployed as VMs) minimizing the computational impact. On one side, the scheduler algorithm has as input a list of tasks ($T1, T2, \ldots$), assigned by the CLS algorithm, with all the nonattended job requests that have been received in the node (see Figure 2). On the other side, each time a computational virtual resource is released in the node, the scheduler is notified and a matching process is done. Then, a resource driven algorithm selects the best candidates to match resources according to some efficient metric. This scheduling is called a *coarse-grained scheduling* because FPGA resources are busy (not available for allocation) until the end of an assigned task.

In addition, in this paper, a multiobjective metric is proposed and computed as the fraction of the computational work of a task and the deadline or time available to finish it up in order to fulfill the SLA.

Therefore, the multiobjective metric measures the computational workload demanded on the node over the time; that is, it provides a measure of the computational stress. And it is a fact that the higher the computational requirements, the more the energy consumption. That is why the task with most demanding requirements is matched to an FPGA resource. In other words, the algorithm matches the most demanding job according to this criterion to the VM with FPGA accelerators.

The computational work parameter involved in the metric reflects the workload that must process the VM. Then, we need to apply some criteria to calculate this value. For example, in our experiments with multimedia applications this parameter is computed as the number of video frames to be processed. Other examples are data-encryption services, where this parameter could be the size of the data that will be encrypted, or social network analysis services, where it could be the number of tweets stored in a log file. Techniques such as application profiling and using data from previous executions are frequently used to estimate this value.

The other parameter used in the metric is the deadline that is established by the client in the SLA.

To sum up, the node level scheduling process works as follows (see Algorithm 1): First, the selected metric is computed for each task. Next, the scheduler sorts all tasks based on that metric and the FPGA is assigned to the most demanding task. This process is repeated every time the FPGA becomes available. To achieve this goal, the status of the FPGA is periodically monitored. This synchronization process is an important issue in the overall scheduling process. Thus, some implementation details will be explained in the next subsection. The algorithm is aware of the heterogeneity of the node, because the most demanding task is executed in a VM deployed with FPGA resources.

5.1. Implementation Details. FPGAs are physically plugged into computing nodes via the PCI express bus. The communication between VMs and the FPGAs is made through the Intel Virtualization Technology for Directed I/O (VT-d) [24], which is an extension of the Intel Virtualization Technology. It allows a direct exchange of data between I/O devices and VMs.

As pointed out above, a strategy able to control the communication between the Node Level Scheduler (NLS) and the VMs currently running in the node is required. So, an FPGA synchronization service has been developed which consists of an exchange of status messages between the NLS and every VM deployed in the node. The full node information is wrapped into a "JSON" [25] data structure because it is easy to parse. The NLS is implemented as a daemon. It is constantly listening to the status information from the VMs currently deployed at the node. Each VM also has a daemon, which is periodically monitoring the status of the FPGA (whether an FPGA is attached to the VM or not). Besides, the VMs periodically send their identification and the progress of the running applications through an UDP socket connection to the NLS. If a VM is using an FPGA, its status is also forwarded to the scheduler by means of updating a "status file."

Figure 3 depicts the process of reassigning the FPGA to different VMs. In this example, the FPGA is initially attached

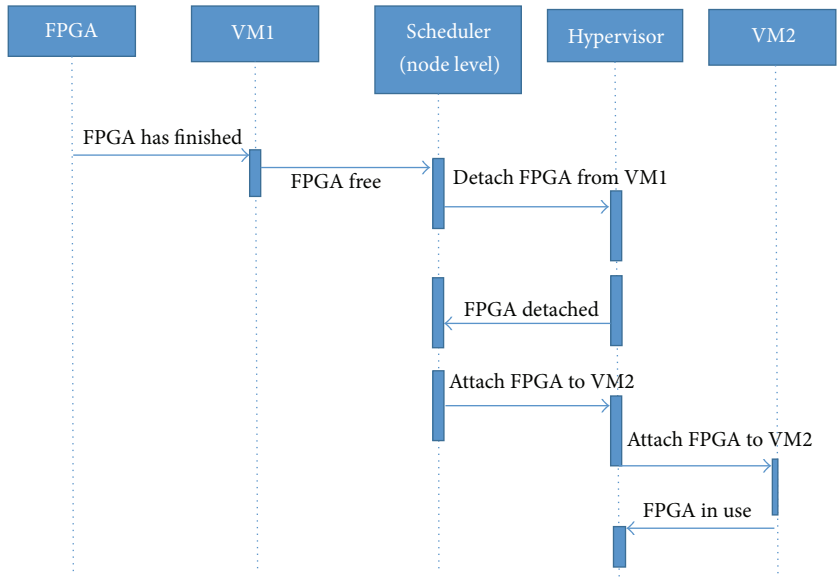


Figure 3: FPGA synchronization service example.

to VM1. Then, when computation on the FPGA ends, VM1 updates the status of the FPGA as "free." As a result, the NLS learns this state and gets aware of the fact that the reassignment process can be made safely. At this point the NLS can detach the FPGA from the current VM and attach it to another VM (i.e., the one in the head of the list of pending requests) through the hypervisor. The VM which finds an FPGA attached to it (VM2, in this case) immediately uses it and updates its status accordingly.

We use KVM [26] as hypervisor. It enables the management of the FPGA because it supports hot-plug devices with VT-d. So, it is a must that the underlying hardware supports VT-d. The NLS uses the "device_add" [27] command to attach the FPGA to a particular VM. Also, when the FPGA has fulfilled a request, the NLS uses "device_del" [27] to release the FPGA.

The NLS is not only focused on the FPGA assignment, it also considers the CPUs. In this case, the NLS matches VMs to physical CPUs using First-Fit criteria. Moreover, the scheduler uses the "taskset" [28] linux command to set the CPU affinity of a running process given its identification (PID). So, each VM has a PID and the CPU affinity "bonds" this process to the first available CPU. This is made in order to avoid swapping when a VM is assigned to different cores along time. Finally, due to the complexity of coding FPGAs, the RIFFA framework offered by Jacobsen and Kastner in [18] has been used to develop the hardware acceleration design.

In Section 7.3 the evaluation of this strategy is shown.

6. Fine-Grained Strategy

In the previous section, the Node Level Scheduler decides to assign the FPGA to a VM and the FPGA is not released until the end of the task running in that VM. But it might be the case that the task does not need to be accelerated during its whole runtime in order to fulfill its deadline. Moreover, it might occur that other tasks (which initially was not placed at the head of the list due to the value of its metric) would depend on the use of the FPGA to fulfill its deadline. Thus, reconsidering the allocation of the FPGA with a finer granularity could improve the number of fulfilled SLAs and, consequently, improve the system ROI. Under this context, a strategy which consists of dividing each task into smaller subtasks (called "chunks") has been explored. A trade-off between the execution time of a chunk and the frequency of checkpoints must be taken into account to decide the size of a chunk. The rationale is to reschedule the FPGA more often, so that more tasks can potentially benefit from the acceleration and power efficiency of the FPGA.

More precisely, all the chunks have the same requirements of computational workload, but different deadline restrictions. Figure 4 shows how a task with size S and deadline D is divided into three chunks, with sizes s_1, s_2, s_3 and deadline D. Chunk size is set according to the characteristics of the application that implements the task. This can be a certain amount of data to be processed or any other application relevant parameter. As an example, for the video processing service, the chunk size is set as a certain number of video frames to process.

Thus, Node Level Scheduler decisions on the use of the FPGA are taken every time and a VM finishes the processing of a chunk (see check point in Figure 4). In the current implementation, this is done every minute. The Node Level Scheduler selects the best candidate VM and attaches the FPGA to it as is depicted in Figure 5. The process is similar as in the previous case but now communication and scheduling are done at the chunk level. It is worth to notice that the application is completely independent and parallelizable. What it means is that the application receives a group of data as input and when the task has been finished the system sends the result as output. For acceleration process a wrapper function is used to map the application that will use CPU and

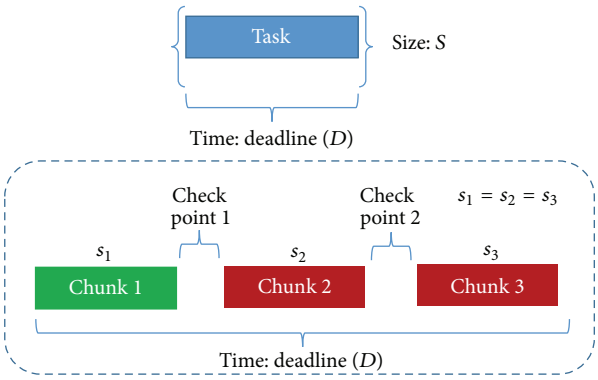


Figure 4: Division of a task to chunks.

FPGA at the same time because it is different than the one who is only running over a CPU. All the VMs of a node can take advantages of the FPGA's features and the whole node can keep the performance while the energy consumption is reduced.

In Section 7.3, some experiments are made showing the necessity to adopt a balanced criteria for all requests due to the fact that FIFO leads to lack of efficiency.

7. Evaluation

To evaluate the impact of the scheduling algorithms proposed in this paper some experiments have been carried out on a Heterogeneous Cloud Computing testbed environment, with a real application case study. Details on the platform, application, and workload are given next, prior to explaining some evaluation results.

All the experiments have been run on a real testbed, depicted in Figure 6.

Hardware is composed of an Intel Core i5 with 6 Gb of DRAM and an FPGA Virtex 6 by Xilinx (a ML605 Evaluation Kit, based on the powerful XC6VLX-240T-1FFG1156 [29]).

Additionally, as a way to measure the energy the testbed includes a power consumption monitor node. More precisely, the WattsUp PRO [30] power meter has been used. This device aims to provide an independent managed and accessed power data collecting mechanism. This device must be positioned between the computing node power supply and the main power plug, as shown in Figure 6. The output data can be recorded into a file according to a predefined interval or when particular events occur (i.e., when the power consumption exceeds a threshold previously configured). In our experiments, the monitoring frequency has been setup to one sample per second.

7.1. Application. As real use case application an Image Convolution Software (ICS) is used as a service provided by cloud. This application consists on convolving an image with a filter or kernel (integer value) in both directions horizontal and vertical. This technique can be applied also to process sequences of video. Different type of filters can be used depending on the target. For our experiments we use a Sobel filter to process a sequence of video. This video is an AVI file with resolution 720 × 384. The application has two versions, the first one runs over a CPU and the another one over a combination between CPU and FPGA.

7.2. Workload. In order to generate a realistic cloud workload we have run the real ICS application and monitored its behavior. Thus, for every test, a bag of tasks (20) was created. Each of these requests are defined by two parameters: the deadline of the task and the number of videos to process (which relates to the amount of data to be processed by the task). There are two types of service requests, depending on how demanding their deadline. The first type of requests is based on more relaxed deadline requirements (referred to as soft requirements, SR), while the second type of requests exhibits a more demanding deadline (referred to as high requirements, HR).

The HR requests are more challenging than SR because they require to fulfill stricter deadlines and specific resources such as FPGAs to ensure the QoS-related SLOs.

To set up the deadline of each service request, we have previously characterized the application. Basically, the Sobel filter application has been run on the cloud using different computational resources for different video sequences. In this step, the number of frames, the power consumption, and the time have been stored. Thus, we have got the profile of each video stream. Then, the deadline is assigned to each input request as a random value between the execution times obtained in the profile with a margin of tolerance. For instance, for a video stream composed of three chunks the deadline could be a random value between 450 and 480 seconds while for nine chunks, the values are between 1400 and 1430.

To emulate the behavior of cloud clients, the input requests rate follows a Poisson distribution $\lambda = 1/t_{\text{interval}}$. The t_{interval} is the arrival time of the next requests and it was tuned for this concrete scenario to avoid an early saturation.

Moreover, three types of bags of tasks have been created, depending on the ratio between SR and HR requests. Note that the same input request rate has been used for the different types of bags of task, only the deadline of the requests has been modified, in order to get a fair comparison of the scheduling algorithms under evaluation. In particular, workloads include 10%, 25%, and 50% of HR requests.

7.3. Evaluation for Coarse-Grained Scheduling. To carry out a performance evaluation of the novel multiobjective metric proposed in Section 5 hereinafter called Highest-Job-First (HJF), QoS compliance and energy consumption are taken into account. Moreover, for the sake of comparison both a random (RND) (with none consideration for QoS requirements) and an Earliest-Deadline-First (EDF) scheduling algorithm (the VM with closest deadline gets the FPGA use, independently of the amount of data to be processed) have also been implemented.

In order to understand the behavior of the whole system we have selected several groups of metrics.

The first one describes the behavior of the whole system. It is composed of the total energy used for the system to process a bag of tasks (energy-to-solution [31]).

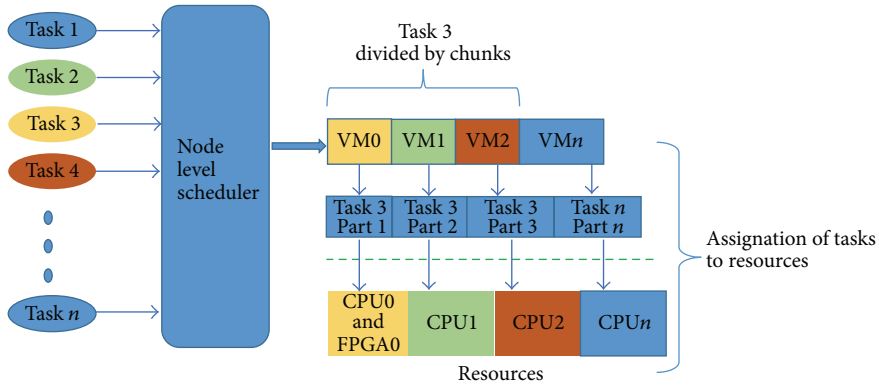


Figure 5: Scheduler by chunks.

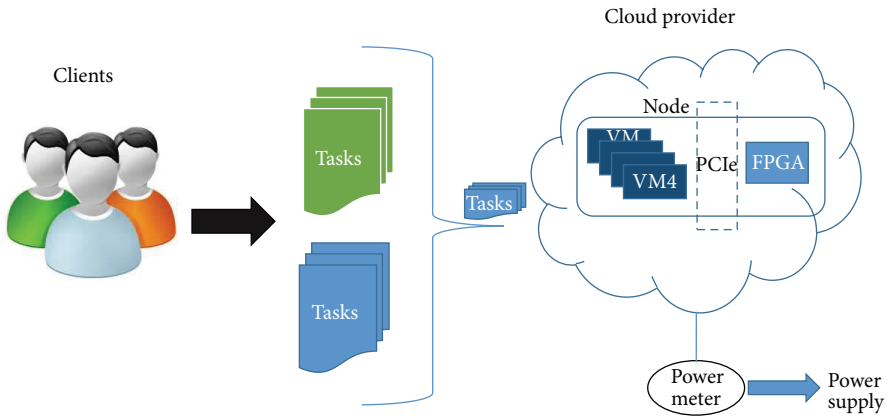


Figure 6: Cloud computing experimental testbed.

Table 1: Energy (KJoules) for different scheduling criteria.

% HR request	FPGA assignation criteria		
	RND	EDF	HJF
10%	648	638	614
25%	653	638	616
50%	657	639	634

Table 2: Percentage of fulfilled SLAs for different scheduling criteria.

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	85%	85%	90%
25%	75%	80%	85%
50%	60%	65%	70%

The second group is relative to SLA compliance, the percentage of SLA fulfilled successfully, and the average energy invested per fulfilled SLA.

And the last one is focused on application performance, namely, the average number of frames successfully processed per unit of time (fps).

Table 1 shows energy consumed by the system. Results show that the HJF and EDF metrics reduce the energy-to-solution for all the type of bags of tasks with respect to the RND baseline algorithm.

Hence, from these results we can point out that the FPGA-aware Node Level Scheduler should use a QoS-aware metric.

Nevertheless, before getting a clue time an energy-to-solution should be analyzed together with the number of SLAs fulfilled. Table 2 shows the percentage of fulfilled SLAs for the different scheduling techniques. Results show that HJF gets more fulfilled SLA for all the input workloads.

Figure 7 shows the cost per SLA measured as the energy consumed. In all the cases, the cost per SLA increases with the percentage of high requirements. These result from the fact that some unlucky scheduling decisions can affect in a quite negative way the future tasks. The allocation of the FPGA to the most demanding task will change the state of the FPGA to nonfree for a period of time. Then, if the physical node receives a quite demanding task over this period of time, as no preemption of task is possible, the new request only can run in CPU resources. This can lead to a nonfulfilled SLA. Finally, regarding the performance as is shown in Figure 8 the amount of frames per second keeps an acceptable level even when more demanding requests are processed. So, in the next

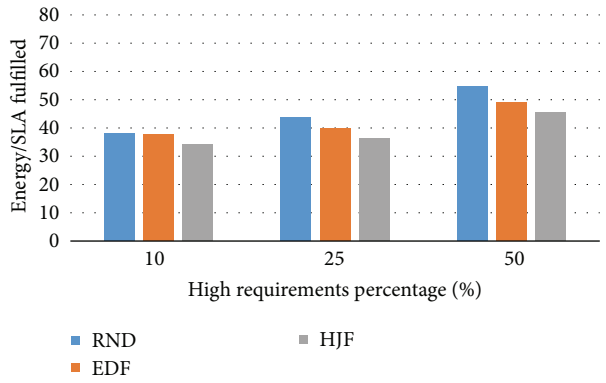


Figure 7: Energy (KJoules) per number of fulfilled SLAs.

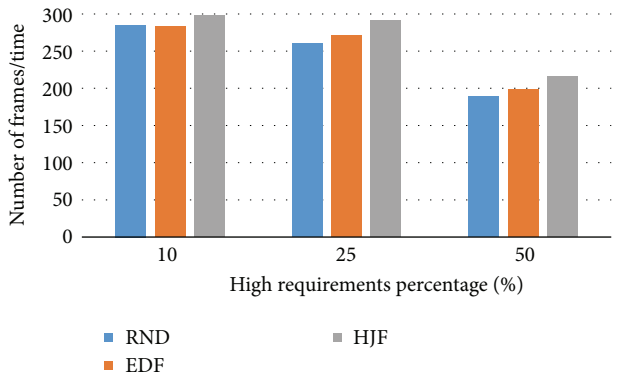


Figure 8: Number of successfully processed frames per second.

section we will evaluate our solution (fine-grained scheduling see Section 6) to address this problem.

7.4. Evaluation for Fine-Grained Scheduling. In this section the fine-grained scheduling strategy proposed in Section 6 will be evaluated. Evaluation conditions are the same as detailed in Section 7. Recall that now the task is subdivided into chunks and the FPGA assignation mechanism is job-aware.

Table 3 shows that the energy tends to rise with the percentage of high requirements requests (these metrics present the same behavior as in previous evaluation). However, the energy per SLA fulfilled decreases 35% for the HJF scheduling strategy in comparison with the other strategies (RND, EDF) (see Figure 9). Results also show that the energy relative to the number of fulfilled SLAs tends to decrease which means a save of effective energy. Thus, the combination of both an FPGA as an accelerator and an efficient scheduling strategy allows the system to save energy. The reason is the system's selection of the most suitable resources for each request and the speedup of the FPGA with lower energy impact. On the other hand, the percentage of fulfilled SLAs (see Table 4) shows a slightly upward trend (20%) for the HJF scheduling with 50% of high requirements. What it means is that the system is able to keep an acceptable ratio of SLA fulfilled.

Figure 10 shows the performance of the system in terms of the number of processed frames per second. The use of

Table 3: Energy (KJoules) for different scheduling criteria (fine-grained).

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	686	665	626
25%	694	699	646
50%	774	772	696

Table 4: Percentage of SLAs fulfilled for different FPGA assignation criteria (fine-grained).

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	90%	95%	95%
25%	80%	80%	85%
50%	55%	70%	75%

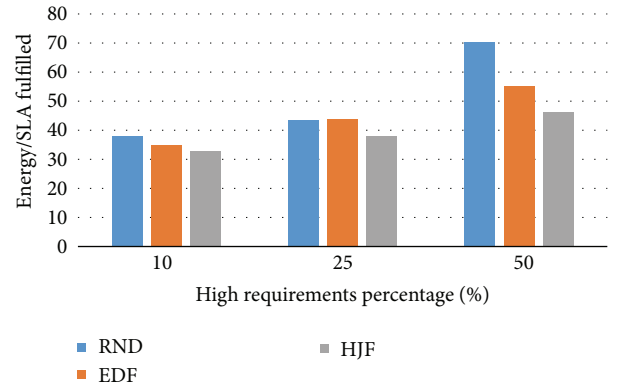


Figure 9: Energy (KJoules) per number of fulfilled SLAs (fine-grained).

the proposed HJF strategy clearly outperforms the RND and obtains similar results to the EDF. However, as shown before, this is achieved with a lower energy consumption for the HJF metric.

7.5. Fine-Grained Scheduling Optimization. The last strategy presented above raises the following concern: when we check with a certain frequency which VM needs most the FPGA in order to fulfill its requirements, why not to check also if the deadline is past? In case the deadline of the task had arrived, the VM would be killed. In this way, no task keeps running after its deadline, savings on energy.

When a SLA is violated, apart from penalties, some amount of energy is wasted because the system uses the resources to run a task that will not successfully end before its deadline. The system behavior shows that keeping VMs alive has a direct effect on the total energy and number of SLA fulfilled. Now, an optimized scheduling, where VMs running a request whose SLA has been violated are killed, is analyzed.

In this scenario, even though the scheduling and communication issues are the same as before, energy consumption for all the FPGA assignation strategies is reduced as shown in Table 5.

Table 5: Energy (KJoules) for different scheduling criteria (optimized fine-grained).

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	678	649	597
25%	638	619	611
50%	637	563	586

Table 6: Percentage of SLAs fulfilled for different FPGA assignation criteria (optimized fine-grained).

% HR requests	FPGA assignation criteria		
	RND	EDF	HJF
10%	90%	95%	100%
25%	80%	85%	90%
50%	65%	70%	85%

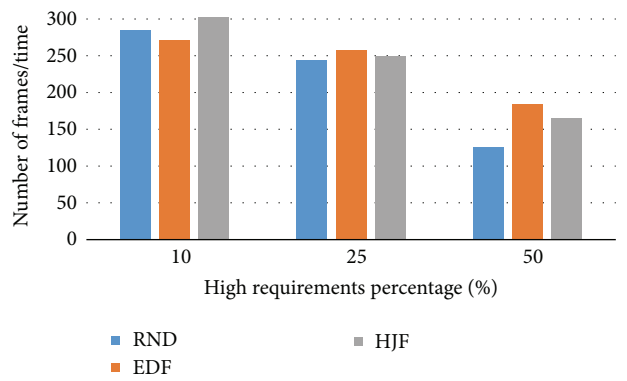


Figure 10: Number of successfully processed frames per second (fine-grained).

On the other hand, if we consider the effective energy per fulfilled SLA, 40% of consumed energy is saved (see Figure 11). In addition, the percentage of successfully fulfilled SLAs is 15% better for HJF (see Table 6). Finally, Figure 12 shows an increase of 20% of frames per second for HJF in the most demanding scenario (50% of high requirements).

To sum up, releasing the resources involved in the execution of a task when its deadline is due improves the percentage of fulfilled SLAs and increases the performance of the cloud system while reducing the energy waste. A comparison summary of the node level scheduling impact is presented in Table 7.

Figure 13 shows the number of frames unsuccessfully processed due to the SLA violation occurring. However, the number of frames unsuccessfully processed decreases significantly for the optimized fine-grained strategy because it allows taking advantage of killing the VMs when their deadline is achieved.

7.6. Evaluation Summary. Table 7 summarizes the different results when applying the different FPGA assignation strategies and scheduling techniques. It can be seen that

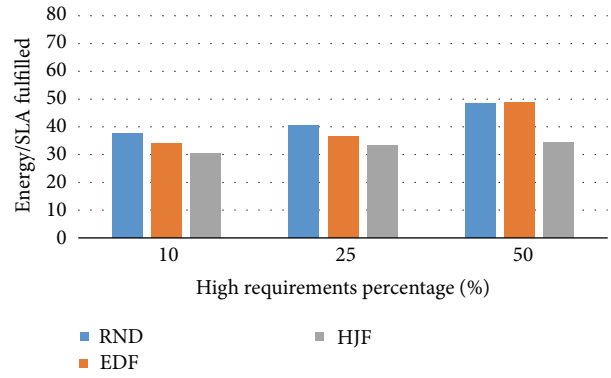


Figure 11: Energy (KJoules) per number of SLAs fulfilled (optimized fine-grained).

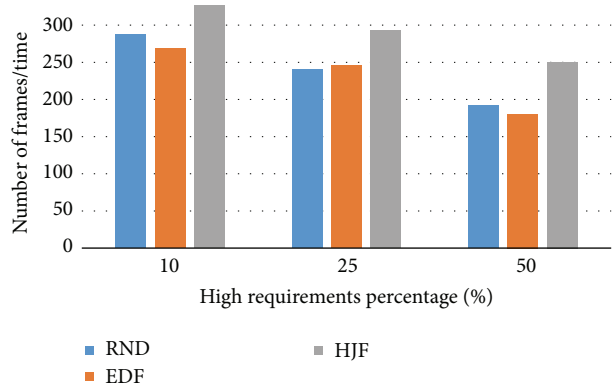


Figure 12: Number of successfully processed frames per second (optimized fine-grained).

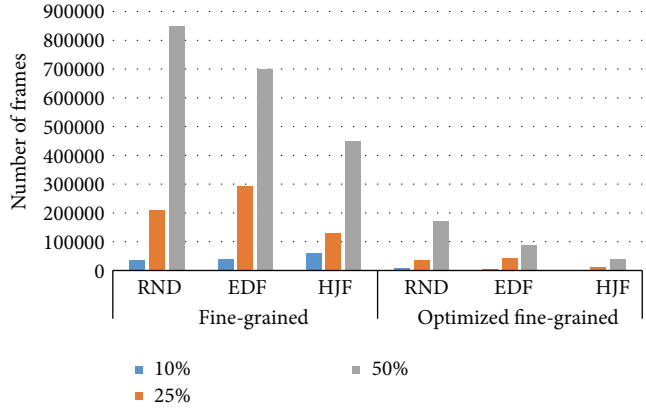


Figure 13: Number of processed frames belonging to unsuccessful SLAs (wasted).

for all the cases, the proposed HJF policy obtains the best performance, in terms of SLA fulfillment and energy consumption. Moreover, the configuration that yields the best results is combining HJF policy with the optimized fine-grained assignation strategy. Recall that for "fulfilled SLAs" and "Frames processed per second," the greater is better, while for "Energy/fulfilled SLA" the lower is better.

Table 7: Summary of tests.

% of high requirements	Scheduling strategy	FPGA assignation criteria	% of fulfilled SLAs	Energy/fulfilled SLA	Frames processed per second
10%	Coarse-grained	RND	85	38	285
		EDF	85	37	283
		HJF	90	34	300
	Fine-grained	RND	90	38	284
		EDF	95	35	271
		HJF	95	32	302
	Optimized fine-grained	RND	90	37	286
		EDF	95	34	268
		HJF	*100*	*30*	*326*
25%	Coarse-grained	RND	75	43	260
		EDF	80	39	271
		HJF	85	36	291
	Fine-grained	RND	80	43	244
		EDF	80	43	257
		HJF	85	38	249
	Optimized fine-grained	RND	80	48	240
		EDF	85	48	245
		HJF	*90*	*34*	*292*
50%	Coarse-grained	RND	60	54	188
		EDF	65	49	198
		HJF	70	45	215
	Fine-grained	RND	55	70	126
		EDF	70	55	184
		HJF	75	46	164
	Optimized fine-grained	RND	65	48	192
		EDF	70	48	180
		HJF	*85*	*34*	*250*

8. Conclusions and Future Work

We have presented a hierarchical scheduler framework to manage heterogeneous resources within a SaaS cloud environment. This framework is responsible for selecting on-demand the most suitable resources for a service while keeping a balance between performance and power consumption. The key of the framework is to maximize the number of fulfilled SLAs saving energy by making efficient use of FPGA devices. Thus, we have proposed a novel dynamic scheduling metric which considers a combination of the workload of a task, its remaining time to complete the task, and the deadline. In addition, a fine-grained accelerator-aware scheduling has been developed and improved by releasing the resources associated with a task as soon as the system realizes its SLA is not going to be fulfilled.

The proposed techniques have been compared with some well-known scheduling strategies such as Earliest-Deadline-First and Random. Experiments carried out over a real testbed indicate that the proposed metric together with the optimized fine-grained scheduling strategy increases the performance of the system even when more demanding requirements are involved. Moreover, these techniques save 23% of the total energy while the percentage of fulfilled SLAs is 85% under the most demanding workload conditions. This can be explained by the fact that FPGAs (as a accelerators) have a great ratio performance/power consumption for certain type of applications. Thus, a proper use of these devices can be turned into benefits for clients and providers.

The efficiency of this approach can be further improved by using more sophisticated strategies such as machine learning algorithms. Also, as a future work we will include the memory and network metrics as variables to optimize the use of the computational resources.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

Acknowledgments

This work was supported by the Spanish Government under Grant TIN2015-66972-C5-2-R (MINECO/FEDER) and by Ecuadorian Government under the SENESCYT Scholarships Project.

References

[1] J. P. Orellana, M. B. Caminero, and C. Carrión, "On the provision of SaaS-level quality of service within heterogeneous private clouds," in *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '14)*, pp. 146–155, IEEE, London, UK, December 2014.

[2] J. Proaño, C. Carrión, and B. Caminero, "Towards a green, QoS-enabled heterogeneous cloud infrastructure," in *Proceedings of the 25th Heterogeneity in Computing Workshop in Conjunction with International Parallel and Distributed Processing Symposium (HCW-IPDPS '16)*, IEEE, Chicago, Ill, USA, May 2016.

[3] PANACEA: Proactive autonomic management of cloud resources, 2016, http://www.panacea-cloud.eu/.

[4] Microsoft Research: Cloud Security & Cryptography, April 2016, http://research.microsoft.com/en-us/projects/cryptocloud/.

[5] The Cloud Standards Wiki, http://cloud-standards.org/.

[6] R. Ranjan, B. Benatallah, S. Dustdar, and M. P. Papazoglou, "Cloud resource orchestration programming: overview, issues, and directions," *IEEE Internet Computing*, vol. 19, no. 5, pp. 46–56, 2015.

[7] Q. Zhang and W. Shi, "Energy-efficient workload placement in enterprise datacenters," *Computer*, vol. 49, no. 2, pp. 46–52, 2016.

[8] T. Guérout, T. Monteil, G. Da Costa, R. Neves Calheiros, R. Buyya, and M. Alexandru, "Energy-aware simulation with DVFS," *Simulation Modelling Practice and Theory*, vol. 39, pp. 76–91, 2013.

[9] Open Compute Project, http://www.opencompute.org/.

[10] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*, Morgan Kaufmann, 1st edition, 2013.

[11] F. Chen, Y. Shan, Y. Zhang et al., "Enabling fpgas in the cloud," in *Proceedings of the 11th ACM International Conference on Computing Frontiers (CF '14)*, Cagliari, Italy, May 2014.

[12] T. Trader, *EU Projects Unite on Heterogeneous ARM-Based Exascale Prototype*, 2016, http://www.hpcwire.com/2016/02/24/eu-projects-unite-exascale-prototype.

[13] A. Putnam, A. M. Caulfield, E. S. Chung et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture (ISCA '14)*, pp. 13–24, IEEE, Minneapolis, Minn, USA, June 2014.

[14] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency Computation Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[15] OpenCL, The Open Standard for Parallel Programming of Heterogeneous Systems, April 2016, http://www.khronos.org/opencl/.

[16] Vivado, Vivado Design Suite, April 2016, http://www.xilinx.com/products/design-tools/vivado/.

[17] J. Auerbach, D. F. Bacon, P. Cheng, and R. Rabbah, "Lime: a java-compatible and synthesizable language for heterogeneous architectures," in *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '10)*, pp. 89–108, Reno, Nev, USA, 2010.

[18] M. Jacobsen and R. Kastner, "RIFFA 2.0: a reusable integration framework for FPGA accelerators," in *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL '13)*, pp. 1–8, IEEE, Porto, Portugal, September 2013.

[19] K. Vipin and S. A. Fahmy, "DyRACT: a partial reconfiguration enabled accelerator and test platform," in *Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL '14)*, pp. 1–7, IEEE, Munich, Germany, September 2014.

[20] C.-W. Tsai and J. J. P. C. Rodrigues, "Metaheuristic scheduling for cloud: a survey," *IEEE Systems Journal*, vol. 8, no. 1, pp. 279–291, 2014.

[21] S. Byma, J. G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "FPGAs in the cloud: booting virtualized hardware accelerators with OpenStack," in *Proceedings of the 22nd IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM '14)*, pp. 109–116, Boston, Mass, USA, May 2014.

[22] OpenStack, Open source software for building private and public clouds, April 2016, https://www.openstack.org/.

[23] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized FPGA accelerators for efficient cloud computing," in *Proceedings of the IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom '15)*, pp. 430–435, Vancouver, Canada, November 2015.

[24] Intel, Intel Virtualization Technology of Directed I/O, Architecture Specification, Rev. 2.2, 2014, http://www.intel.com/content/www/us/en/embedded/technology/virtualization/vt-directed-io-spec.html.

[25] D. Crockford, "The application/JSON media type for JavaScript object notation (JSON)," 2006.

[26] KVM, Kernel Based Virtual Machine (KVM), 2008, http://www.linux-kvm.org/.

[27] WeidongHan, How to assign devices with vt-d in kvm, 2009, http://www.linux-kvm.org/index.php?title=How_to_assign_devices_with_VT-d_in_KVM&action=info.

[28] R. M. Love, Taskset, 2004, http://linux.die.net/man/1/taskset.

[29] Xilinx ML605, Virtex-6 FPGA ML605 Evaluation Kit, March 2016, http://www.xilinx.com/publications/prod_mktg/ml605_product_brief.pdf.

[30] PowerMeterStore, Power Meter Store, March 2016, http://www.powermeterstore.com/p1206/watts_up_pro.php.

[31] R. da Rosa Righi, C. A. da Costa, V. F. Rodrigues, and G. Rostirolla, "Joint-analysis of performance and energy consumption when enabling cloud elasticity for synchronous HPC applications," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 5, pp. 1548–1571, 2016.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 4708195, 10 pages
http://dx.doi.org/10.1155/2016/4708195

Research Article

ANCS: Achieving QoS through Dynamic Allocation of Network Resources in Virtualized Clouds

Cheol-Ho Hong, Kyungwoon Lee, Hyunchan Park, and Chuck Yoo

Korea University, 145 Anam-ro, Seongbuk-gu, Seoul 02841, Republic of Korea

Correspondence should be addressed to Chuck Yoo; chuckyoo@os.korea.ac.kr

Received 27 February 2016; Accepted 19 May 2016

Academic Editor: Zhihui Du

To meet the various requirements of cloud computing users, research on guaranteeing Quality of Service (QoS) is gaining widespread attention in the field of cloud computing. However, as cloud computing platforms adopt virtualization as an enabling technology, it becomes challenging to distribute system resources to each user according to the diverse requirements. Although ample research has been conducted in order to meet QoS requirements, the proposed solutions lack simultaneous support for multiple policies, degrade the aggregated throughput of network resources, and incur CPU overhead. In this paper, we propose a new mechanism, called ANCS (*Advanced Network Credit Scheduler*), to guarantee QoS through dynamic allocation of network resources in virtualization. To meet the various network demands of cloud users, ANCS aims to concurrently provide multiple performance policies; these include weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation. In addition, ANCS develops an efficient work-conserving scheduling method for maximizing network resource utilization. Finally, ANCS can achieve low CPU overhead via its lightweight design, which is important for practical deployment.

1. Introduction

The use of cloud computing technology is rapidly increasing, because it can provide computing resources to remote users efficiently in terms of time and cost. By using cloud platforms, cloud users can run diverse applications without considering the arrangement of the hardware platforms. In addition, because of the agile and elastic traits of cloud computing, the amount of computing resources can be adjusted to reflect the requirements of each user. Therefore, cloud computing is rapidly being disseminated for general-purpose, network and database server, and high-performance computing applications [1].

System virtualization [2] is an enabling technology of cloud computing. A hypervisor or virtual machine monitor (VMM) provides cloud users an illusion of running their own operating system (OS) on a physical platform. Using a hypervisor in the basement of the software layer, cloud vendors can realize IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Service as a Service), which are core services of cloud computing. Recent representative hypervisor titles for cloud computing include KVM [3], Xen [4], and VMware ESXi [5].

To satisfy the diverse requirements of cloud users, cloud computing providers have recently allowed users to select the Quality of Service (QoS) of each resource. As a result, users can specify the needs appropriate to their programs and pay fees according to the QoS level. However, it is challenging for cloud providers to support stable QoS for applications in each virtual machine (VM) because of interference between cloud users [6]. In such environments, users can experience unpredictable performance and performance degradation. Therefore, research on achieving performance isolation during resource sharing is gaining significant attention [6–10].

In particular, guaranteeing network performance is widely studied because network performance influences the Quality of Experience (QoE) that defines the satisfaction level of each user. Most previous research efforts [11–14] focused on methods to dynamically adjust network performance. However, the proposed methods have limitations in that (1) they can support only one type of policy among several viable ones and (2) they incur high CPU overhead by adopting feedback control, which frequently monitors the

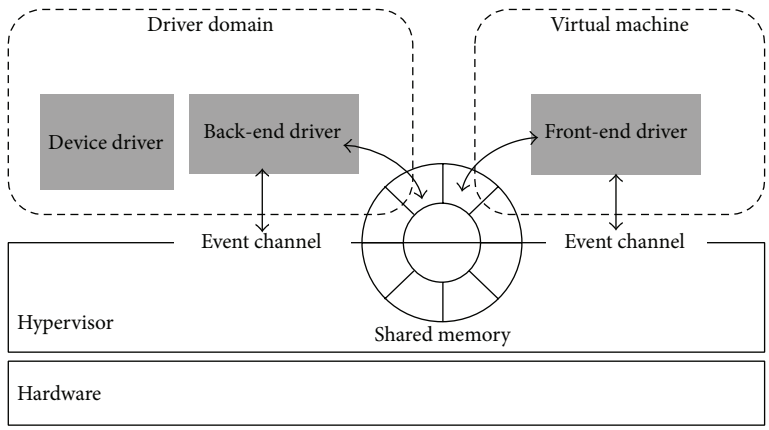


Figure 1: Xen I/O architecture.

actual usage of the network and compensates insufficient resources. Therefore, a new approach is required to meet the demands of the users while significantly reducing overhead.

In this paper, we propose ANCS (*Advanced Network Credit Scheduler*), which can dynamically provide network performance according to the requirements of each user of a virtualized system. Compared to solutions proposed in the previous research, ANCS can support several network performance control policies simultaneously; these include weight-based proportional sharing, minimum bandwidth guarantees, and maximum bandwidth guarantees. In addition, at all times ANCS provides a work-conserving scheme [15] that distributes the unused resources of any VM to other VMs, which maximizes total network utilization. Finally, ANCS incurs low CPU overhead through its lightweight design.

The main contributions of this paper related to previous studies are as follows:

(i) We deliver the details of ANCS that can satisfy diverse performance demands through multiple performance policies in virtualization. Each cloud computing tenant has different performance requirements. In order to satisfy the needs, ANCS provides three performance policies: weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation. We elaborate on how to guarantee desired performance according to each demand.

(ii) We develop an efficient work-conserving method for utilizing network resources. In a non-work-conserving method, when a VM cannot fully utilize an allocated resource, the unused amount is wasted, resulting in decreased overall network utilization. To maximize the utilization, ANCS develops an efficient work-conserving method to guarantee high utilization regardless of the resource usage of each VM.

(iii) We design ANCS to use minimal CPU resources when managing the network performance of VMs. We do not utilize packet inspection or packet queuing to reduce overhead in the QoS control.

The remainder of this paper is structured as follows: In Section 2, we explain the background of virtualization and the methods it uses to distribute network resources. Section 3 elaborates on the design of ANCS. Section 4 shows the performance evaluation results. Section 5 explains related work. Finally, we present our conclusions in Section 6.

2. Background and Motivation

We select Xen [4] for implementing ANCS because it is an open-source hypervisor and also supports a simple round robin scheduling method. We can regard this scheduling method as a baseline for our experiment and intend to improve the scheduling method in terms of supporting weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation.

2.1. System Virtualization and Xen. System virtualization allows several OSs to be consolidated on a single physical machine. At the bottom of the system software stacks, the hypervisor multiplexes all system resources, including processors, memory, and I/O devices; it exports the virtualized resources in the form of a VM. Therefore, each VM can provide a complete system environment (composed of virtual CPUs, virtual memory, and virtual devices) to each guest OS.

Xen [4] is an open-source hypervisor that adopts a paravirtualization technique to minimize virtualization cost. The paravirtualization technique enables communication between the hypervisor and a guest OS by providing a communication channel, which is referred to as a hypercall. Hypercalls are analogous to system calls between an OS and a user application. Hypercalls request Xen to execute sensitive instructions on behalf of the guest OS. They include processor state update operations such as memory management unit (MMU) updates and physical interrupt masking operations, which guest OSs cannot execute directly. To contain hypercalls, the source code of a guest OS must be modified to facilitate paravirtualization.

2.2. The I/O Model of Xen. To process I/O requests from guest OSs, Xen adopts the back-end driver in domain 0, which is a driver domain, and the front-end driver in the guest OS [16], as depicted in Figure 1. The back-end driver delivers

I/O requests using the shared memory from the front-end drivers to the actual device driver in domain 0. Afterward, the back-end driver conveys the responses to each front-end driver by notifying them via virtual interrupts. The front-end driver behaves as an actual device driver in the guest OS. It receives I/O requests from the guest OS and delivers them to the back-end driver. It also brings the processed result to the guest OS. For network devices, the front-end network driver creates virtual network interfaces called *vif*s. The virtual network interfaces behave as the actual network devices in the guest OS. In systems that have several VMs, the back-end network driver processes the requests of several virtual network interfaces in a round-robin manner without a proportional sharing policy. Therefore, each competing guest OS will have the same network performance, which cannot satisfy the different requirements.

3. Design of ANCS

In this section, we present the details of ANCS, an advanced network credit scheduler that dynamically allocates network resources to VMs for guaranteeing the network performance of VMs under various scenarios. First, we describe the design goals of ANCS that aim to guarantee QoS for network virtualization. Next, we elaborate on the coarse-grained algorithm of ANCS using its flow chart. Finally, we explain the fine-grained subalgorithms of ANCS for achieving diverse performance goals.

3.1. Design Goals. The goals of ANCS that aim to implement a network scheduler for guaranteeing QoS in virtualization are as follows.

(i) Multiple Performance Policies. ANCS aims to meet the network demands of each VM, which are described by multiple performance policies. Recent cloud computing users tend to run diverse network applications on their VMs. These applications have different performance demands, especially in terms of networks; minimum sustainable bandwidth is an example of such a requirement. In order to meet the various needs, ANCS provides the following three performance policies: weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation. ANCS aims to satisfy the desired performance of cloud users by allowing them to select an appropriate performance policy among the three policies, according to their demands.

(ii) High Network Resource Utilization. ANCS endeavors to efficiently utilize the total network resources of a system by applying a work-conserving method. In a non-work-conserving environment, when a VM receives network resources in proportion to its weight and does not consume its allocated amount, the unused network resources then become wasted, resulting in decreased total network utilization. In order to enhance resource management efficiency, ANCS adopts a work-conserving method, which yields unused network resources to other VMs that have pending network requests. This maximizes the resource utilization of the entire system regardless of the usage of each VM.

(iii) Low CPU Overhead. ANCS focuses on minimizing additional CPU overhead in achieving QoS for networks. Other research efforts that aimed to guarantee QoS failed to reduce CPU overhead, because of the complexity of the proposed algorithms [12, 14, 17]. These studies adopted either packet inspection or packet queuing for throttling the sending packet rate per network interface, and therefore this mechanism demands nonnegligible computation resources. Unfortunately, this causes total network performance to deteriorate. ANCS adopts a credit-based simple accounting algorithm that incurs only minimal overhead, which prevents performance degradation during the QoS control.

Although ANCS is implemented in the back-end driver of the Xen hypervisor, ANCS can be applied to other hypervisors that use a paravirtual model where the back-end and the front-end cooperate with each other to deal with network packets. In a paravirtual model, each hypervisor processes packets in a similar manner. For example, the back-end receives/sends packets from/to the front-end by using shared memory composed of descriptor rings. Also, after receiving and sending packets, the back-end delivers virtual interrupts to the front-end in order to notify that the packets have been processed. Because of this architectural similarity, we believe that ANCS can still satisfy the performance requirements in other hypervisors.

3.2. ANCS Algorithm. In basic terms, ANCS is based on the proportional share scheduling mechanism that allocates network resources to each virtual interface in proportion to its weight. Each virtual interface, called *vif*, is owned by its VM and behaves like an actual network interface within the VM. ANCS operates in the driver domain of the Xen hypervisor, particularly in the back-end driver that plays the role of the communication channel between the hardware device driver and VMs, as depicted in Figure 2. ANCS processes the network operation requests of VMs in a round-robin manner, checking whether a *vif* has sufficient resource allocation to possess the network resources. For this purpose, ANCS uses the credit concept [18, 19] to represent the amount of resource allocation. While network resources are being used, every *vif* consumes its credit according to the requested size; the credit value of a *vif* is recharged regularly in proportion to its weight. If the credit value that a *vif* has is less than the requested size, ANCS does not process its requests. The *vif* must then wait for the next credit value to be allocated. Therefore, the amount of credit a *vif* has determines the network bandwidth of the corresponding VM. The amount of credits allocated is calculated by the configured performance policy of each VM.

Figure 3 depicts an overall flow chart of the ANCS algorithm. In this section, we elaborate on the coarse-grained algorithm of ANCS using the flow chart. We introduce variables used in the ANCS algorithm and provide a general description of them. Then, we explain the fine-grained subalgorithms of ANCS. The detailed operations of ANCS will be described in Sections 3.3 and 3.4.

The accounting function of ANCS runs every 30 ms and allocates network resources to *vif*s on a physical machine in order. The value of 30 ms is selected to decrease the

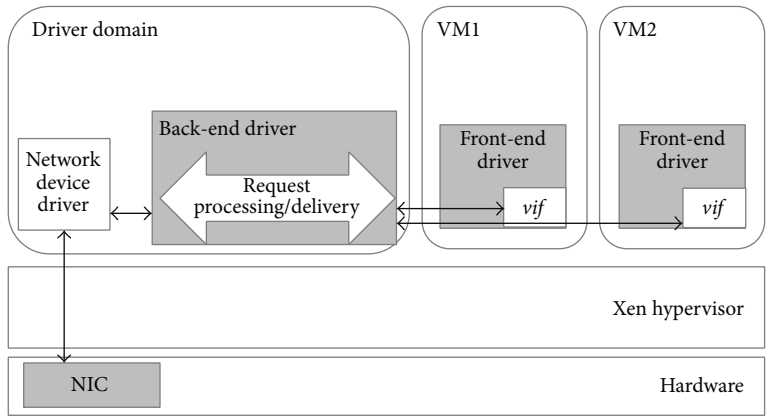


Figure 2: Virtualized system architecture with ANCS.

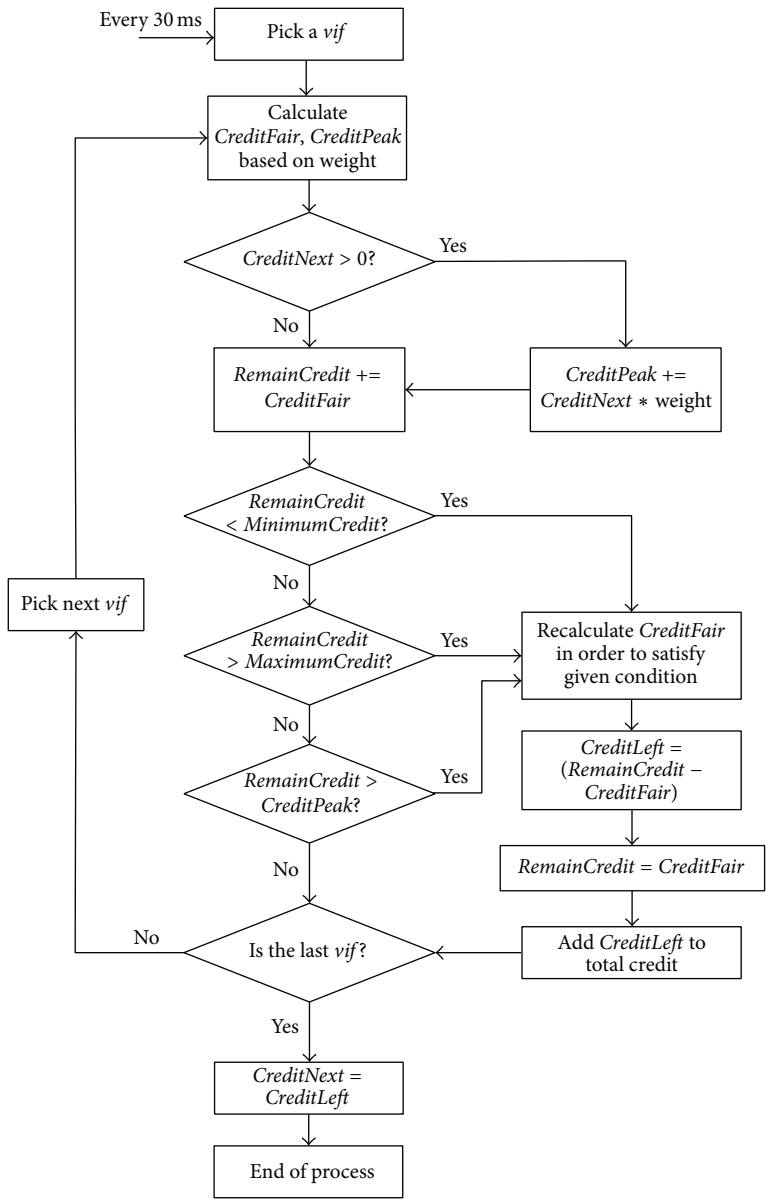


Figure 3: ANCS scheduling algorithm.

overhead of periodically running the accounting function. Please note that regardless of this period each virtual interface can execute if it has sufficient credit values to process its network packets. Therefore, the 30 ms period does not affect the network latency of each virtual interface. The following procedure is performed per *vif*.

First, the two variables required to obtain the fair share of each *vif* are calculated: *CreditFair* and *CreditPeak*. *CreditFair* is determined depending on the weight of a *vif*. *CreditFair* represents the fair share of network resources for a *vif* according to its weight. Then, *CreditPeak* is calculated. *CreditPeak* indicates the amount of available resources for a *vif* based on the assumption that the *vif* dominates the network resource when other *vif*s are idle. *CreditPeak* is used to maintain the total amount of credits in the system in order to distribute the proper amount of credits to each *vif*.

Second, the procedure to support work-conserving and various performance policies is performed. ANCS checks whether *CreditNext* is greater than zero. *CreditNext* is accumulated in the previous scheduling period when some *vif*s did not fully use their credit values. A positive *CreditNext* value indicates that there were unused credits in the previous scheduling period. This value is added to *CreditPeak* in order to distribute unused credits in the current scheduling period. We will explain this in more detail in Section 3.4. Then, for proportional sharing, ANCS adds *CreditFair* to *RemainCredit*, which is the current credit value of the *vif*. In order to support minimum bandwidth reservation and maximum bandwidth limitation, ANCS determines whether the credits allocated to the *vif* satisfy the configured performance policy, which will be explained in depth in Section 3.3.

Finally, if the current *vif* is the last *vif* in the system, the ANCS process for a single scheduling period is finished. If there are additional *vif*s to which credits can be allocated, ANCS selects the next *vif* and iterates the credit calculation. If a VM does not consume its allocated credits in the current period, ANCS adds the remaining credit value to *CreditNext* to give other VMs a chance to consume unused credits.

3.3. Multiple Performance Policies. For the diverse performance requirements of cloud computing users, ANCS provides multiple performance policies, including weight-based proportional sharing, minimum bandwidth reservation, and maximum bandwidth limitation. Weight-based proportional sharing is a base policy that proportionally differentiates the amount of allocated resources based on the weight of each VM. Minimum bandwidth reservation guarantees that the quantitative network performance of a VM is always greater than the configured value, *MinimumCredit*, as shown in Figure 3. When the amount of credits allocated to a *vif* becomes less than *MinimumCredit*, ANCS adjusts *CreditFair* to satisfy the minimum bandwidth; the *CreditFair* value of the *vif* is set to *MinimumCredit* in order to support minimum bandwidth. When supporting minimum bandwidth, the aggregated amounts of minimum bandwidth requests from all *vif*s should not exceed the total throughput of the physical device. By using the minimum bandwidth reservation, ANCS can prevent performance degradation of a specific VM and guarantee sustainable minimum bandwidth. On the other hand, maximum bandwidth limitation prevents aggressive resource consumption by a specific VM. If the resources allocated to a *vif* exceed the *MaximumCredit* value, ANCS reclaims surplus resources from the *vif* for distribution to other VMs.

The administrator can apply an appropriate performance policy to each VM according to the performance demands of each user. The designated policy can be dynamically changed during runtime. In addition, multiple performance policies can be applied in a conjunctive form. For instance, when a user requires a specific boundary of network performance (e.g., larger than 200 Mbps and less than 500 Mbps), both the reservation and limitation requirements can be applied by specifying the minimum and maximum bandwidth values. Moreover, ANCS can satisfy the different performance demands of multiple applications in a single VM, because the designated performance policy is based on a *vif*, not a VM. Therefore, a VM with several *vif*s can establish various network performance requirements by assigning a different performance policy to each *vif*.

3.4. Support for Work-Conserving. ANCS supports work-conserving, in which a *vif* is idled when it does not have network requests or responses. Work-conserving is a crucial factor in cloud computing; it allows resources to be allocated efficiently and system utilization to be maximized. For work-conserving, ANCS modifies the amount of allocated credits according to fluctuations in network usage. When a VM does not fully use its credits, ANCS gives other VMs a chance to receive unused credits by distributing additional credits to them.

When a *vif* is idled for a certain amount of time, the *RemainCredit* value of the *vif* can exceed *CreditPeak*, which denotes the maximum credit value a *vif* can have. Then, the credits of this *vif* must be distributed to other *vif*s. For this purpose, *CreditLeft*, which is obtained by subtracting *CreditFair* from *RemainCredit*, is added to the total credit value. This increases the credits that can be allocated to the next *vif*s. If a *vif* exceeds its *CreditPeak*, ANCS resets the credit value of the *vif* to *CreditFair*, to give it a credit value according to its weight. By distributing idle resources to active VMs, ANCS can efficiently utilize network resources of the entire system.

4. Evaluation

For evaluation, we implement ANCS in the driver domain, domain 0, of the Xen hypervisor. The Linux version of domain 0 is 3.10.55, and the version of Xen is 4.2.1. The experiments are conducted on two identical physical servers, each of which has an Intel i7 Quad core 3770 CPU running at 3.5 GHz, 16 GB of RAM, and a Gigabit Ethernet card. One of them runs the Xen hypervisor with our modified driver domain, and the other one runs vanilla Linux 3.10.11 working as an evaluation machine. Because ANCS does not require any modification on guest OSs, we run unmodified vanilla Linux 3.2.1 for each VM. Iperf [20] benchmark is used to measure the network performance of each VM. In addition,

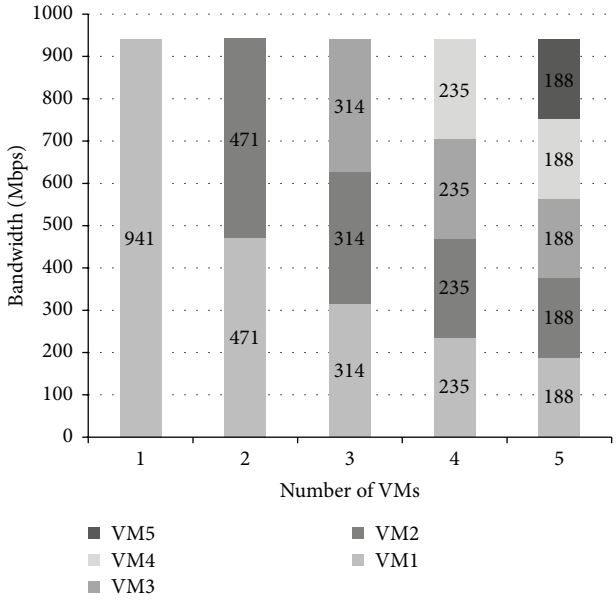


Figure 4: Bandwidth of VMs without ANCS.

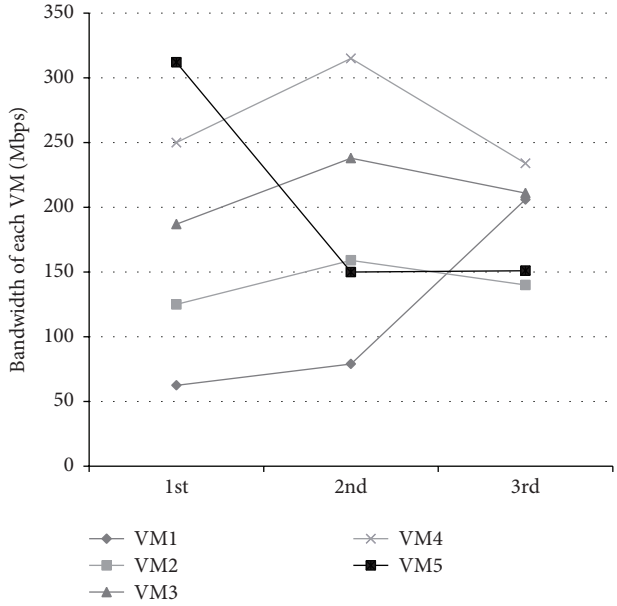


Figure 5: Bandwidth of VMs with different performance policy settings.

we use Xentop to measure the CPU utilization of the driver domain, which is described in Section 4.4.

4.1. Base Performance. Before we show the evaluation results of ANCS, we produce the base performance for comparison. For this purpose, we measure the network performance of default VMs on the Xen hypervisor without ANCS. We ran Iperf while launching up to five VMs one by one. As depicted in Figure 4, the network performance of each VM on our system is always even. When there is only one VM, the VM (VM1) consumes all network resources, achieving 941 Mbps bandwidth. As the number of VMs increases, the system's network resources are equally shared among VMs. This occurs because the unmodified driver of the Xen hypervisor processes the requests of several virtual network interfaces in a round-robin manner with the same weight. Therefore, each guest OS has the same network performance.

4.2. Multiple Policies. To demonstrate that ANCS guarantees the network performance of VMs under various scenarios, we change the performance policy during runtime and observe that the change is reflected properly in the system. When the system is started, we set the weights of five VMs for weight-based proportional sharing as {VM1 = 1/15, VM2 = 2/15, VM3 = 3/15, VM4 = 4/15, VM5 = 5/15}; thus, the ratio of the weights is 1 : 2 : 3 : 4 : 5. After measuring the performance of each VM, we additionally set the maximum bandwidth limitation policy to VM5. The bandwidth of VM5 is set to 150 Mbps while the weight-based proportional sharing is maintained. Finally, the minimum bandwidth reservation is set to VM1 with a value of 200 Mbps. Similarly, the previous configuration applied to other VMs is maintained.

The results of the experiment are shown in Figure 5. At the beginning of the experiment, ANCS allocates credits to *vif*s according to their weights. Therefore, the network performance of each VM is differentiated proportionally, as expected. When the maximum bandwidth limitation policy is applied to VM5 in the second phase, VM5 shows the bandwidth limited to 150 Mbps. As the credit value allocated to VM5 decreases compared to the first phase, the network performance of other VMs increases naturally. Moreover, the increment in the bandwidth of other VMs is proportional to their weights. During the last phase, the bandwidth of VM1 increases to 200 Mbps because of the reservation policy, while the bandwidth of VM5 is preserved as 150 Mbps, the same as it was in the second phase. Then, VM2, VM3, and VM4 receive less credit value compared to the second phase, and therefore their performance decreases.

Based on the above experiment, ANCS shows that it guarantees the network performance of VMs under various scenarios that combine the policies explained in Section 3.3. The scenarios include all three performance policies provided by ANCS. Our evaluation demonstrates that ANCS is capable of handling various performance requirements in cloud computing.

4.3. Support for Work-Conserving. In cloud computing, efficient resource management is crucial for reducing maintenance costs. For better efficiency in resource management, ANCS maximizes resource utilization even if the number of VMs and their performance policies are changed. We evaluated the efficiency of ANCS in resource management by measuring aggregated bandwidth in different environments.

First, Figure 6 provides the bandwidth results of the VMs on a physical server. We configured only one policy (weight-based proportional sharing) on all VMs. In addition, we maintained the weight of each VM for weight-based proportional sharing as {VM1 = 1/15, VM2 = 2/15, VM3 = 1/5, VM4 = 4/15, VM5 = 1/3}, throughout the remaining experiments.

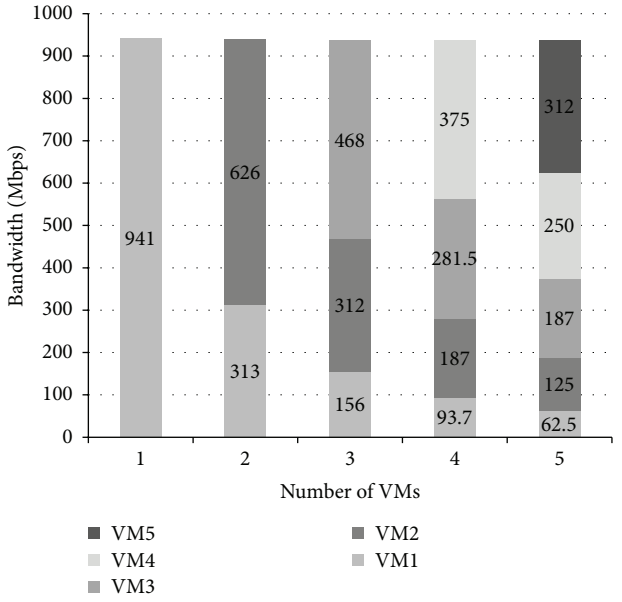


Figure 6: Aggregated bandwidth of VMs with ANCS (only weight-based proportional sharing is applied).

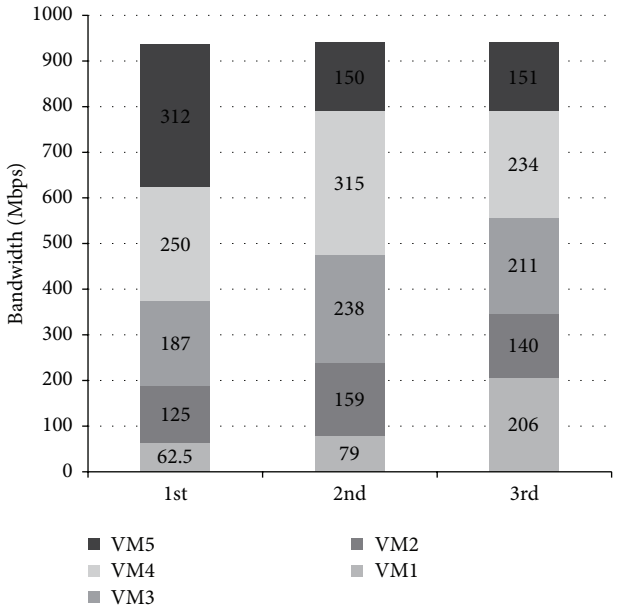


Figure 7: Aggregated bandwidth of VMs with ANCS.

We then launched the VMs sequentially (VM1 through VM5) and measured the network performance of each VM. Evaluation results show that the network performance of each VM is proportional to the weight. While the network performance of each VM changes according to the weight, the aggregated bandwidth maintains 940 Mbps. We find that ANCS fully supports work-conserving scheduling by achieving the maximum aggregated bandwidth, independent of the number of VMs on the physical server.

Next, Figure 7 shows the aggregated bandwidth in the three experiment scenarios described in Section 4.2. Through this experiment, we aimed to show that ANCS maximizes resource utilization even if the performance policy of each VM is changed. In the first phase, only weight-based proportional sharing is applied to all VMs. In the second phase, the maximum limitation policy is applied to VM5 while maintaining the weight-based proportional sharing of the other VMs. Finally, the minimum bandwidth reservation is set to VM1. In the final phase, all of the performance policies that ANCS provides are applied on the system: reservation to VM1 (200 Mbps), limitation to VM5 (150 Mbps), and weight-based proportional sharing to all VMs. As depicted in Figure 7, the aggregated bandwidth always achieves the maximum bandwidth, 940 Mbps, in all the experiment scenarios. It is clear that the work-conserving property of ANCS is not affected by the required performance policy of each VM, or by the number of VMs.

4.4. CPU Overhead. In cloud computing, it is important to decrease CPU overhead in terms of energy efficiency when applying a new policy to the system. To illustrate the CPU overhead caused by ANCS, we measured the CPU utilization in the driver domain while sequentially generating VMs one by one. Figure 8 shows the CPU utilization in three cases: Native Xen, Linux traffic controller, and ANCS. Both Native Xen and the traffic controller are based on the Xen hypervisor; they run unmodified Linux for driver domains. In the experiment with Native Xen, the network performance of each VM is equal because the default setting of Xen is fair sharing. For the traffic controller experiment, we utilize the Linux traffic controller in the driver domain to differentiate the network performance of VMs, similar to ANCS. The traffic controller does not support work-conserving and only provides network performance policies with fixed performance values; this is different from ANCS,

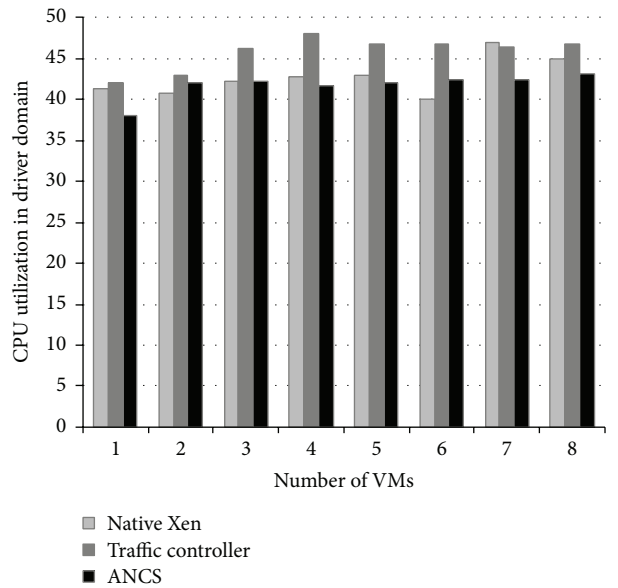


Figure 8: CPU utilization in the driver domain when the number of VMs is increased from one to eight.

Table 1: Latency values in ms measured by a ping program when the number of VMs is increased from one to five.

	Number of VMs				
	1	2	3	4	5
Native Xen	0.30	0.34	0.25	0.30	0.32
Traffic controller	0.30	0.30	0.30	0.32	0.32
ANCS	0.24	0.28	0.23	0.23	0.34

Table 2: Latency values in ms measured by Netperf when the number of VMs is increased from one to five.

	Number of VMs				
	1	2	3	4	5
Native Xen	0.12	0.13	0.12	0.12	0.12
Traffic controller	0.12	0.12	0.12	0.13	0.13
ANCS	0.12	0.12	0.12	0.12	0.12

which dynamically adjusts the network performance of VMs. As depicted in Figure 8, the traffic controller demands non-negligible computation resources as it adopts packet queuing for throttling the sending packet rate. As ANCS adopts a credit-based simple accounting algorithm that incurs only minimal overhead, ANCS shows low CPU utilization in the driver domain, which is comparable to the two different techniques.

4.5. Network Latency. In this section, we evaluate network latency in ANCS and compare the results with Native Xen and the Linux traffic controller in the same experiment setup described in Section 4.4. For the performance policy, we configure the weight of each VM as one to provide the same bandwidth to VMs, which prevents different resource allocation from affecting network latency. We use both a ping program and Netperf [21] with TCP_RR to quantify the latency of UDP and TCP pings, respectively. The ping program sends 64-byte UDP packets to a client whereas Netperf with TCP_RR sends 1-byte TCP packets to a client. For these experiments, we respectively execute both programs between a client and a VM for 60 seconds and take the average value, because network latency fluctuates on each measurement. Tables 1 and 2 show the latency values (in ms) of each method measured by the ping program and Netperf, respectively, when the number of VMs is increased from one to five. In both results, ANCS shows lower or similar latency compared to Native Xen and the traffic controller. As explained in Section 3.2, although the accounting function of ANCS runs every 30 ms, virtual interfaces with sufficient credit values can run at certain points regardless of this period. Therefore, the 30 ms period does not affect the network latency of each virtual interface.

5. Related Work

As cloud computing based on virtualization becomes prevalent and the requirements of each user are diversified, techniques for improving QoS are increasingly studied. In particular, network devices are considered to offer unstable QoS, because sharing of the devices involves another scheduling layer (such as the scheduler in the back-end driver).

vSuit [11] adopts a feedback controller in order to analyze variations in network performance and to react to such deviations by adjusting network resources to each guest OS. Using this mechanism, vSuit can offer maximum bandwidth reservation and limitation for each VM. Furthermore, CPU overhead is shown to be under 1%. However, this research does not support weight-based proportional sharing, and the experiments are performed in 100 Mbps environment.

DMVL [12] provides a technique to distribute the network bandwidth to each VM in a stable and fair manner. For this purpose, it separates the logical data path and the request I/O queue belonging to each VM. In addition, it records the allocated and consumed amounts of network bandwidth for each VM by using shared memory. This information is used to adjust the amount of network bandwidth allocated in the next iteration. The limitation of this research is that this technique incurs nonnegligible CPU overhead in domain 0, up to 7%.

PSD [13] differentiates the network performance of each VM based on each virtual network interface. It schedules each virtual interface by means of a leaky bucket controller and a time slot-based resource allocator, which distribute the resource in proportion to a different ratio. However, this approach cannot provide absolute bandwidth assignment.

The Linux traffic controller [14, 17] is a traditional approach to adjusting the network bandwidth of each network application. It classifies each packet according to its header information and configures different bandwidths according to each classification. In the paravirtualized Xen, the traffic controller in domain 0 can be used to perform network performance differentiation by classifying the packets based on the IP address of each VM. In this case, severe CPU utilization occurs (up to 12%) when the number of VMs increases, as shown in Figure 8.

Recently, Silo [22] proposes to consider both VM placement and end host pacing to guarantee message latency in cloud computing. In Silo, message latency between two VMs is determined by the capacity of network switches connecting two virtual machines. End host pacing controls the transmission rate in a server by utilizing hierarchical token bucket scheduling. Even though Silo guarantees quantitative network latency in cloud computing, it only covers communication between two specific VMs. When a VM starts new connection with another VM or migrates to a different server, Silo needs to recalculate guaranteed latency.

Cerebro [23] predicts bounds on the response time performance of web APIs exported by applications hosted in a PaaS cloud. However, Cerebro only predicts response time of cloud applications and does not involve network scheduling in the case of violation of predicted response time. Moreover, Cerebro incurs computational overheads since it adopts a static analysis to predict the response time.

As shown in Table 3, the solutions proposed to enhance network QoS either provide only a single policy or incur high CPU overhead through the use of high-frequency feedback controllers. ANCS can support various policies, including proportional sharing and minimum/maximum network bandwidth reservation. It also maximizes the total

Table 3: Comparison of virtual network scheduling methods.

	ANCS	vSuit	DMVL	PSD	Traffic controller	Silo	Cerebro
Work-conserving	o	o	o	x	o	x	x
Proportional sharing	o	x	o	o	o	x	x
Minimum and maximum reservation	o	o	x	x	x	o	x
CPU overhead	Low	Low	High	N/A	High	Low	High

network bandwidth by supporting work-conserving while reducing CPU overhead as much as possible.

6. Conclusion

In this paper we propose ANCS, which dynamically enhances network performance to meet the various requirements of users in cloud computing environments. Compared to solutions proposed in previous research efforts, ANCS can provide several network performance control policies, including weight-based proportional sharing, minimum bandwidth guarantees, and maximum bandwidth guarantees, while achieving low CPU overhead.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea Government (MSIP) (no. R0126-16-1066, (SW StarLab) Next Generation Cloud Infra-Software toward the Guarantee of Performance and Security SLA) and Grant no. B0126-16-1046 (Research of Network Virtualization Platform and Service for SDN 2.0 Realization). This work was also supported by Korea University Grant.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] J. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*, Elsevier, New York, NY, USA, 2005.

[3] I. Habib, "Virtualization with KVM," *Linux Journal*, vol. 2008, no. 166, article 8, 2008.

[4] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[5] C. Chaubal, "The architecture of VMware ESXi," VMware White Paper, 2008.

[6] X. Pu, L. Liu, Y. Mei et al., "Who is your neighbor: net I/O performance interference in virtualized clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 3, pp. 314–329, 2013.

[7] H. Park, S. Yoo, C.-H. Hong, and C. Yoo, "Storage SLA guarantee with novel SSD I/O scheduler in virtualized data centers," *IEEE Transactions on Parallel and Distributed Systems*, 2015.

[8] J. Hwang, C. Hong, and H. Suh, "Dynamic inbound rate adjustment scheme for virtualized cloud data centers," *IEICE Transactions on Information and Systems*, vol. E99.D, no. 3, pp. 760–762, 2016.

[9] N. Jain and J. Lakshmi, "PriDyn: enabling differentiated I/O services in cloud using dynamic priorities," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 212–224, 2015.

[10] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: plan when you can," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, pp. 407–420, ACM, 2015.

[11] F. Dan, W. Xiaojing, Z. Wei, T. Wei, and L. Jingning, "VSuit: QoS-oriented scheduler in network virtualization," in *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '12)*, pp. 423–428, Fukuoka, Japan, March 2012.

[12] H. Tan, L. Huang, Z. He, Y. Lu, and X. He, "DMVL: an I/O bandwidth dynamic allocation method for virtual networks," *Journal of Network and Computer Applications*, vol. 39, no. 1, pp. 104–116, 2014.

[13] S. Lee, H. Kim, J. Ahn, K. Sung, and J. Park, "Provisioning service differentiation for virtualized network devices," in *Proceedings of the the International Conference on Networking and Services (ICNS '11)*, pp. 152–156, 2011.

[14] W. Xiaojing, Y. Wei, W. Haowei, D. Linjie, and Z. Chi, "Evaluation of traffic control in virtual environment," in *Proceedings of the 11th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES '12)*, pp. 332–335, Guilin, China, October 2012.

[15] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in Xen," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 2, pp. 42–51, 2007.

[16] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe hardware access with the Xen virtual machine monitor," in *Proceedings of the 1st Workshop on Operating System and Architectural Support for the on Demand IT Infrastructure (OASIS '04)*, pp. 3–7, October 2004.

[17] W. Almesberger, "Linux network traffic control-implementation overview," in *Proceedings of the 5th Annual Linux Expo LCA-CONF- 1999-012*, pp. 153–164, Raleigh, NC, USA, 1999.

[18] K. Mathew, P. Kulkarni, and V. Apte, "Network bandwidth configuration tool for Xen virtual machines," in *Proceedings of the 2nd International Conference on COMmunication Systems and NETworks (COMSNETS '10)*, pp. 1–2, IEEE, Bangalore, India, January 2010.

[19] C.-H. Hong, Y.-P. Kim, H. Park, and C. Yoo, "Synchronization support for parallel applications in virtualized clouds," *The Journal of Supercomputing*, 2015.

[20] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The TCP/UDP bandwidth measurement tool," http://software.es.net/iperf/.

[21] R. Jones, *NetPerf: A Network Performance Benchmark*, Information Networks Division, Hewlett-Packard Company, 1996.

[22] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: predictable message latency in the cloud," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 435–448, ACM, 2015.

[23] H. Jayathilaka, C. Krintz, and R. Wolski, "Response time service level agreements for cloud-hosted web applications," in *Proceedings of the 6th ACM Symposium on Cloud Computing*, pp. 315–328, ACM, Kohala Coast, Hawaii, August 2015.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 5475964, 11 pages
http://dx.doi.org/10.1155/2016/5475964

Research Article

Mining Software Repositories for Automatic Interface Recommendation

Xiaobing Sun,[1,2] Bin Li,[1,2] Yucong Duan,[3] Wei Shi,[1] and Xiangyue Liu[1]

[1]*School of Information Engineering, Yangzhou University, Yangzhou 225127, China*
[2]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China*
[3]*Hainan University, Haikou 570228, China*

Correspondence should be addressed to Xiaobing Sun; sundomore@163.com

Received 30 January 2016; Revised 3 May 2016; Accepted 18 May 2016

Academic Editor: Laurence T. Yang

There are a large number of open source projects in software repositories for developers to reuse. During software development and maintenance, developers can leverage good interfaces in these open source projects and establish the framework of the new project quickly when reusing interfaces in these open source projects. However, if developers want to reuse them, they need to read a lot of code files and learn which interfaces can be reused. To help developers better take advantage of the available interfaces used in software repositories, we previously proposed an approach to automatically recommend interfaces by mining existing open source projects in the software repositories. We mainly used the LDA (Latent Dirichlet Allocation) topic model to construct the Feature-Interface Graph for each software project and recommended the interfaces based on the Feature-Interface Graph. In this paper, we improve our previous approach by clustering the recommending interfaces on the Feature-Interface Graph, which can recommend more accurate interfaces for developers to reuse. We evaluate the effectiveness of the improved approach and the results show that the improved approach can be more efficient to recommend more accurate interfaces for reuse over our previous work.

1. Introduction

Interfaces are an integral part of software projects, which is also an important means for software design. During software development, most of the software projects use interfaces to construct the frame of a project [1–3]. Program interface is an effective way to construct the framework of a new project. Reusing interfaces in software projects can save development time and reduce development cost. Moreover, most of these open source projects are of high quality [4–8]. If we can reuse the interfaces in these open software projects to help develop a new project, it can effectively allow developers to write less code and improve the efficiency of their project development [9–12].

Today, many open source software repositories are available to developers such as SourceForge (https://sourceforge.net/), Git (https://git-scm.com/), and OSChina (http://www.oschina.net/). These software repositories provide an amount of useful information to developers such as coding style, document, and software interfaces. Hence, mining existing software repositories can help developers to identify some interfaces for reuse. One simple and direct method is to check these code files one by one and then extract the files useful for developers. However, it is difficult, costly, and labor-intensive for developers to implement interfaces all by themselves for a new project.

The main problem is how we can deal with these useful information easily. A number of techniques have been proposed to analyze the software repositories for reuse in software development. Chan et al. proposed a code search approach, which returns a graph of API methods [13]. They used a greedy subgraph search algorithm to identify a connected subgraph containing nodes with high textual similarity to the query phrases. Thung et al. proposed a library recommendation approach based on a set of libraries that a project currently uses, which recommends relevant libraries [14, 15]. They proposed a hybrid approach that combines association rule mining and collaborative filtering to recommend libraries for reuse. In addition, they also proposed an API recommendation approach [15]. The APIs are recommended

based on the measure of the relevance by looking into the similarity between description of feature request and description of APIs. However, the search range of these techniques is too large and the process of matching is complex. Moreover, few of them focus on interface recommendation.

Considering the above issues, an automated approach is needed to recommend interfaces to developers more effectively. In our previous work, we proposed an approach to effectively mine the software repositories for automatic interface recommendation [16]. The approach used the LDA (Latent Dirichlet Allocation) topic model [17] to construct the Feature-Interface Graph (FIG) for each project to recommend interfaces based on their usage in each project. However, the number of recommended interfaces is too large, and we only consider the individual open software project for recommendation. But in practice, different projects have different considerations; that is, some interfaces are more important in some projects while some others are less in other projects. In addition, the time to construct the Feature-Interface Graph (FIG) for each project is too large, and developers need to wait a long time to reuse these recommended interfaces. In this paper, we improve our previous work considering these two aspects in the following way. On the one hand, we use the LDA topic model to extract the topics of open source projects in the open source software repositories and identify the relevant open source projects. In this way, we do not need to construct the FIGs for each project in the software repositories, which is costly. On the other hand, we cluster the interfaces of these relevant projects based on the FIG and recommend the interfaces on the clustering results. In this way, the interfaces are recommended based on their general usage in all of the relevant software projects in software repositories. In this paper, we make the following contributions:

(i) We improve our previous work and propose a more effective approach for automatic interface recommendation by mining software repositories.

(ii) We conduct an empirical study to show the performance improvement of our approach, and the empirical results show that the performance of our approach is greatly improved over our previous work.

(iii) We conduct an empirical study to show the accuracy improvement of our approach, and the empirical results show that the accuracy of the recommended interfaces is also greatly improved over our previous work.

The rest of the paper is organized as follows. In Section 2, we introduce the background related to the LDA topic model. We present our approach in Section 3. In Section 4, we conduct an empirical study to evaluate our approach. In Section 5, we discuss the related work. Finally, Section 6 concludes our approach and discusses some future work.

2. Background

Topic models were originated from the field of information retrieval (IR) to index, search, and cluster a large amount of unstructured and unlabeled documents. A topic is a collection of terms that cooccur frequently in the documents of the corpus. One of the mostly used topic models in software engineering community is Latent Dirichlet Allocation (LDA) [17]. It has been successfully and widely applied to analyze the software engineering data to support various software engineering activities [18–22].

LDA is a probabilistic generative topic model for collections of discrete data such as text corpora [17]. It has been applied to information retrieval to automatically organize and provide structure to a text corpus [23]. In LDA, there is a set of topics to describe the entire corpus. Each document can contain more than one of these topics, and each term in the entire repository can be contained in more than one of these topics. Hence, LDA is able to discover a set of ideas or themes that well describe the entire corpus. In this paper, we extract the topics from the software by using the LDA topic model. These extracted topics help us understand the features of the target software.

The basic main idea of LDA is that documents are represented as random mixture over latent topics, where each topic is characterized by a distribution over words [17]. Specifically, the LDA model consists of the following building blocks:

(1) A word is the basic unit of discrete data, defined to be an item from a software vocabulary $V = \{w_1, w_2, \ldots, w_v\}$, such as an identifier or a word from a comment.

(2) A document is a sequence of words denoted by $d = \{w_1, w_2, \ldots, w_n\}$, where w_i is the ith word in the sequence.

(3) A corpus is a collection of documents denoted by $D = \{d_1, d_2, \ldots, d_m\}$.

Given m documents containing k topics expressed over v unique words, the distribution of ith topic t_i over v words and the distribution of jth document over k topics can be represented.

By applying the LDA model, it outputs the distribution values of the topics of the software projects. We use the distribution values to show the accuracy of the topic. More details of LDA can refer to the work of Blei et al. [17].

3. Our Approach

In this section, we present our approach about how to automatically recommend interfaces. To develop a new software, one of the first tasks is to specify the function modules based on requirement analysis. During this process, interface definition is an important task. After specifying the function modules and providing the features about the new program, some suitable interfaces should be defined. If some of the interfaces can be recommended for developers to reuse, it will improve the efficiency of software development and increase the opportunity of code reuse.

The process of our approach is shown in Figure 1. The input of our approach includes the software repositories and the feature request of a new project. We first use the LDA topic model to extract the software projects from software

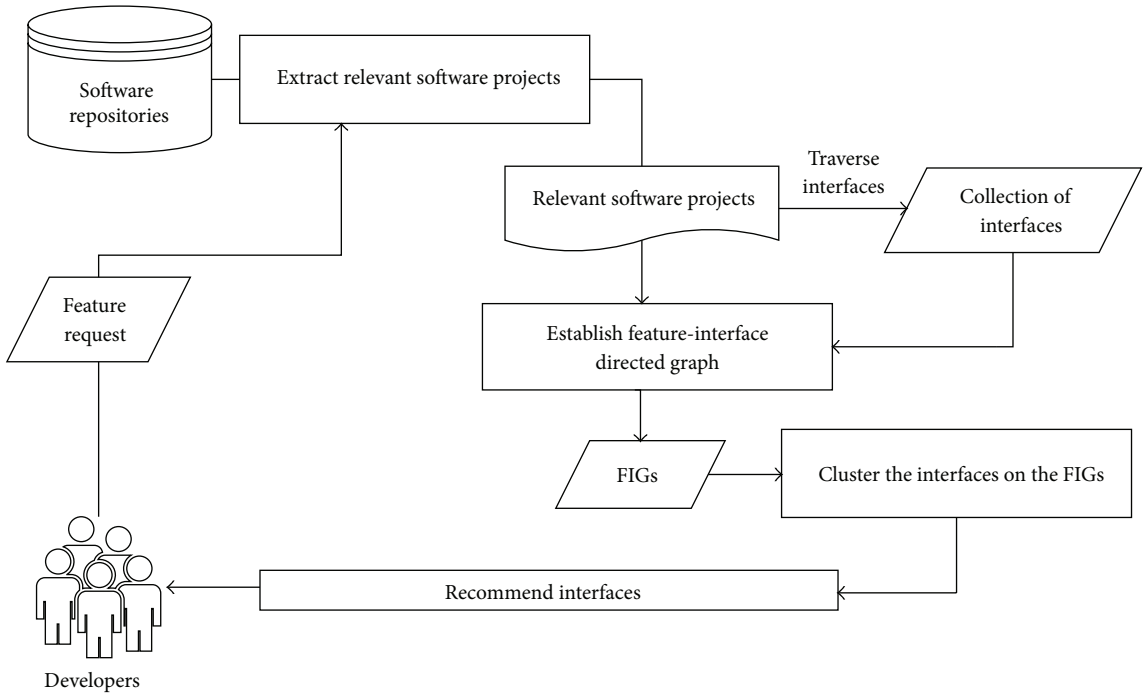


Figure 1: Process of our approach.

Table 1: The topics and the allocation values of topics extracted by LDA from JEdit project.

Topic	Allocation	Word
Topic 0	0.2722	textarea, jedit, gnu, general, software, version, line, copy, color, foundation
Topic 1	0.1836	jedit, entry, code, path, view, url, tag, gjt, jar, title
Topic 2	0.2748	type, undo, scan, scanpos, stream, read, input, encoding, dimision, font
Topic 3	0.2401	jedit, view, box, options, actionhandler, cons, selected, borderlayout, attribute, entity
Topic 4	0.2088	start, end, line, selection, offset, match, input, caret, search, strtline
Topic 5	0.1740	keyevent, vk, key, jedit, length, text, shortcut, installer, historymodel, swing
Topic 6	0.1700	path, vfs, browser, file, jedit, view, directory, session, menu, component
Topic 7	0.2342	offset, line, length, file, start, seg, header, jedit, number, rule
Topic 8	0.2379	interpreter, callstack, file, type, primitive, pat, lgpl, version, notice, item
Topic 9	0.6456	jj, active4, 3r, jjtn, xsp, active4, active2, jjtree, active1, active0

repositories that are relevant to the feature request provided by developers. Then, we traverse the interfaces of these relevant software projects and construct the Feature-Interface Graphs (FIGs) for each of the relevant project. Finally, based on the interfaces on the FIGs, we cluster the interfaces and recommend the candidate interfaces based on the number of their invocations in software repositories.

3.1. Identify Relevant Software Projects. First, we use the topic model (LDA) to extract the topics for each project in software repositories. After extracting the topics, we get some words to describe each open source project. Before using the LDA, we should preprocess these software projects to reduce noise and improve the data quality for use. There are several typical natural language processing (NLP) preprocessing operations for the unstructured source code. We tokenize each word based on common naming practices, such as camel case (*oneTwo*) and underscores (*one_two*), remove common English language stop words (*the*), and strip away syntax and programming language keywords (*public*). More details of the preprocessing operations can refer to our previous work [24]. After preprocessing, we can use the LDA to extract the topics for each project. We see each project as one document collection. For each project, we get K topics to describe the features and K topic allocation values, we denote these topics as a collection named F_i. Table 1 shows an example of the topics extracted from JEdit (http://www.jedit.org/), a famous text editor open source software. There are ten topics extracted from this project. The first column is the topics, the second the allocation of each topic which describes the significance of this topic, and the last column the words describing the topic.

At the beginning of the development process, developers may provide a functional description of their request. The description words of the new software are marked as

a collection. For the feature request of a project, it is a short textual description. So tokenization of the feature request is conducted. During the process of tokenization, the feature request text is turned into tokens. Then, we also get a set of words to represent the feature request; that is, Feature = $\{w_1, w_2, \ldots, w_k\}$. Then, we turn to match the words in the collection with the topics of each software project; that is, $\text{Software}_i = \{w_1, w_2, \ldots, w_j\}$. Specifically, for each project in the software repository, we compute the similarity value (Sim_i) by measuring how many words of the feature request are also present in the LDA representation of a software (Software_i), as shown as the following formula:

$$\text{Sim}_i = \frac{\left|\text{Feature} \cap \text{Software}_i\right|}{\text{Software}_i}. \quad (1)$$

The greater value the similarity (Sim_i) is, the higher the similarity between feature request of the new project and the target project in software repositories is. If the value of the counter satisfies a threshold value, we claim that the target project (Software_i) is relevant to the new project at hand.

3.2. Feature-Interface Graph Construction. In this paper, Feature-Interface Graph (FIG) is an important representation for interface recommendation. In this subsection, we discuss the definition of FIG and the process of how to construct FIG.

FIG is formally defined as follows.

Definition 1 (Feature-Interface Graph). The Feature-Interface Graph (FIG) is a graph; FIG = $\langle N, E, W \rangle$. $N = N_1 \cup N_2$, where N_1 is the topic nodes and N_2 is the interface nodes corresponding to the topic nodes. E is the relation between topic nodes and interface nodes, which is identified by the LDA. W is the weight on the edge, which represents the number of interfaces been invoked in the software.

So on the FIG, there are two types of nodes, that is, topic nodes and interface nodes. Topic nodes are used to store the topics of the software. Interface nodes are the nodes of interfaces used in the software. We distinguish topic nodes and interface nodes because when developers need interfaces to construct the frame of a new project, they need to first provide the feature request about this new project. Then, we can retrieve the interface nodes through matching the topic nodes with the description of the new project.

Then, we traverse each project file and search interfaces used in this project. We store the interfaces in the form of *Project Name-Package Name-File Name-Interface Name*. For example, we find an interface named *BufferChangeListener*. We record its information and store this interface in the form of *jedit-org.gjt.sp.jedit.buffer-BufferChangeListener*. To show the importance of the interfaces, we count the number of interfaces being invoked. Table 2 shows the interfaces in the JEdit and the number of these interfaces being invoked.

After extracting the topics from the relevant projects and traversing the interfaces, their FIGs can be constructed. We mark the collection of topics as the topic nodes, the collection of interfaces as interface nodes, and the number of interfaces being invoked as the weight of edges. Figure 2 shows an example of the FIG for the *JEdit* project. In Figure 2, there are many relevant open source projects in software repositories. For each project, it has its own FIG and has ten topics to describe its features and functions. In our approach, we choose some of these topics to be considered as the most suitable description of *JEdit*. For each project, we also search the interfaces used in corresponding projects and calculate the number of them being invoked. In Figure 2, the weight of edges represents the number of interfaces being invoked.

3.3. Interface Clustering and Recommendation. After constructing the FIGs for each of the relevant projects, we can obtain the information about the topics of the software and their interfaces and the number of each interface being invoked in the software. In our previous work, the interfaces are recommended based on the number of their being invoked for each project. However, different projects have their individual characteristics and the usage of interfaces is different. If we only consider the interface usage in individual project, the recommending results may be inaccurate. So in this paper, the interfaces are recommended based on the whole perspective of interface usage in all of the relevant software projects.

We cluster the interfaces also based on the LDA topic model. Each interface on the FIGs is considered as a file for LDA input. The process of interface clustering includes the following steps:

(1) Natural language processing (NLP) techniques are used to preprocess the interfaces to reduce the noise in the interface data.

(2) LDA is applied to generate the topics.

(3) Clusters are generated, and interfaces having similar topics should be allocated in a cluster.

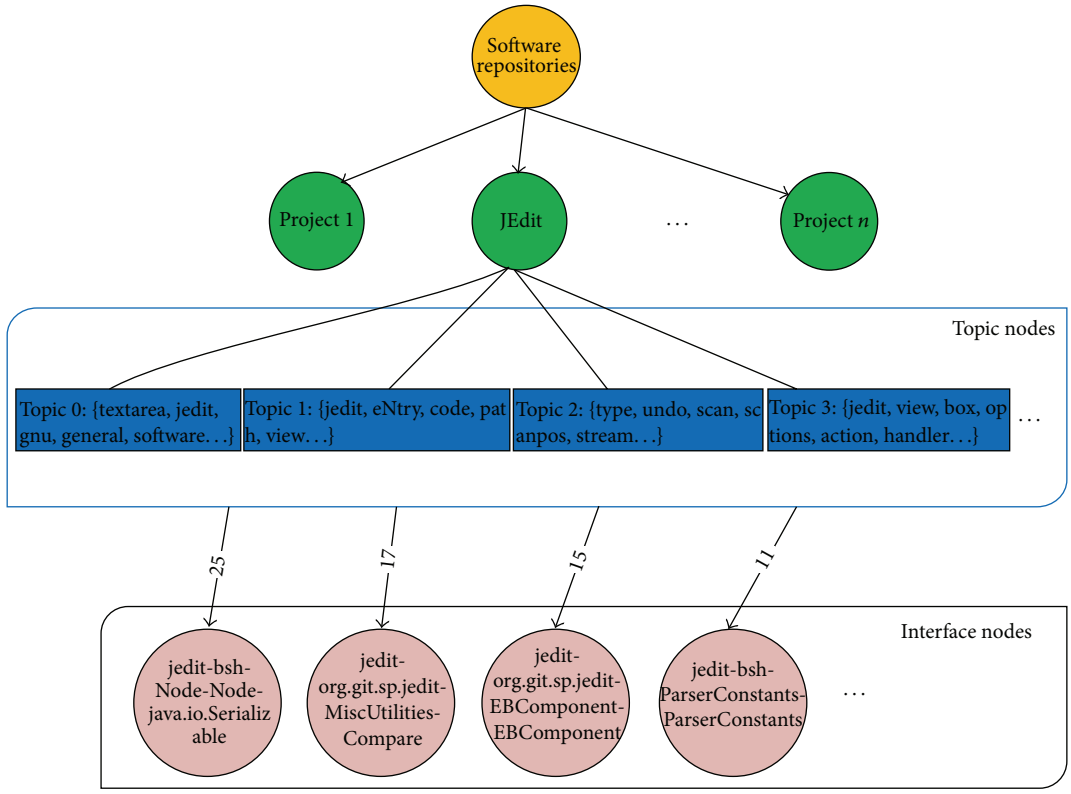
Based on the interface clustering, we can compute the total count of the interfaces being invoked in each cluster. The greater value the count is, the higher the interface in that cluster relevant to a topic is concerned. When this step is finished, the interfaces are ranked and recommended based on the following formula:

$$\text{Rank}\left(\text{Inter}_i\right) = \text{Invoke}\left(\text{Inter}_i\right) \times \frac{\text{Cluster}\left(\text{Inter}_i\right)}{\text{Max}\left(\text{Clustering}\right)}. \quad (2)$$

In (2), $\text{Invoke}(\text{Inter}_i)$ represents the number of Interface i being invoked in a project; $\text{Cluster}(\text{Inter}_i)$ represents the size of the cluster Interface i belonging to. LDA generates K topics. For each topic, it is a cluster, which is composed of some interfaces related to this topic. So when using LDA, we need to set the number of generated topics, K. In addition, in Formula (2), Max(Clustering) represents the largest size of the cluster in the generated clusters. Based on this metric, the top K interfaces can be recommended. According to the directed edges they points to, the interfaces can be automatically determined. Then, we go back to the corresponding project to find the code snippet of the interfaces and recommend the source code to the developers.

Table 2: The interfaces and the number of their being invoked in the JEdit project.

Number	Interface	Invoked account
1	jedit-org.gjt.sp.jedit.browser-BrowserListener-BrowserListener	0
2	jedit-org.gjt.sp.jedit.buffer-BufferChangeListener-BufferChangeListener	1
3	jedit-org.git.sp.jedit.buffer-BufferListener-BufferListener	3
4	jedit-org.git.sp.jedit.gui-DefaultFocusComponent-DefaultFocusComponent	2
5	jedit-org.git.sp.jedit.gui-DockableWindowContainer-DockableWindowContainer	2
6	jedit-org.git.sp.jedit.gui-MutableListModel-MutableListModel	2
7	jedit-org.git.sp.jedit.help-HelpHistoryModelListener-HelpHistoryModelListener	1
8	jedit-org.git.sp.jedit.help-HelpViewerInterface-HelpViewerInterface	1
9	jedit-org.gjt.sp.jedit.indent-IndentAction-IndentAction	5
10	jedit-org.gjt.sp.jedit.indent-IndentRule-IndentRule	2
11	jedit-org.gjt.sp.jedit.menu-DynamicMenuProvider-DynamicMenuProvider	7
12	jedit-org.gjt.sp.jedit.search-HyperSearchNode-HyperSearchNode	2
13	jedit-org.gjt.sp.jedit.search-HyperSearchResults-ResultVisitor	0
14	jedit-org.gjt.sp.jedit.search-HyperSearchTreeNodeCallback-HyperSearchTreeNodeCallback	2
15	jedit-org.gjt.sp.jedit.search-SearchFileSet-SearchFileSet	2
16	jedit-org.gjt.sp.jedit.syntax-TokenHandler-TokenHandler	2
17	jedit-org.gjt.sp.jedit.textarea-ScrollListener-ScrollListener	2
18	jedit-org.gjt.sp.jedit.textarea-ScrollListener-java.util.EventListener	1
19	jedit-org.gjt.sp.jedit.textarea-StatusListener-StatusListener	2
20	jedit-org.gjt.sp.jedit.textarea-StructureMatcher-StructureMatcher	0
21	jedit-org.gjt.sp.jedit-EBComponent-EBComponent	15
22	jedit-org.gjt.sp.jedit-MiscUtilities-Compare	17
23	jedit-org.gjt.sp.jedit-OptionPane-OptionPane	1
24	jedit-org.gjt.sp.jedit-Registers-Register	2
25	jedit-org.gjt.sp.util-ProgressObserver-ProgressObserver	3
26	jedit-org.gjt.sp.util-WorkThreadProgressListener-WorkThreadProgressListener	0
27	jedit-org.objectweb.asm-ClassVisitor-ClassVisitor	1
28	jedit-org.objectweb.asm-CodeVisitor-CodeVisitor	1
29	jedit-org.objectweb.asm-Constants-Constants	1
30	jdeit-bsh-BSHBlock-NodeFilter	1
31	jedit-bsh-BshClassManager-Listener	2
32	jedit-bsh-BshIterator-BshIterator	2
33	jedit-bsh-ConsoleInterface-ConsoleInterface	1
34	jedit-bsh-NameSource-NameSource	0
35	jedit-bsh-Node-Node	1
36	jedit-bsh-Node-Node-java.io.Serializable	25
37	jedit-bsh-ParserConstants-ParserConstants	11
38	jedit-bsh-ParserTreeConstants-ParserTreeConstants	1
39	jedit-com.microstar.xml-XmlHandler-XmlHandler	1
40	jedit-gnu.regexp-CharIndexed-CharIndexed	6
41	jedit-installer-BZip2Constants-BZip2Constants	2
42	jedit-installer-Progress-Progress	2
43	jedit-installer-TarInputStream-EntryFactory	1
44	jedit-java.swing.border-Border	2
45	jedit-java.swing.text.Postion	1
46	jedit-javax.swing.ListModel	1
47	jedit-java.lang.reflect.InvocationHandler	1
48	jedit-java.util.Enumeration	1
49	jedit-java.util.EventListener-EventListener	0



Green node: name of project
Blue rectangle: topics of JEdit extracted by topic model
Pink node: interfaces in JEdit
Weight of edges: number of interfaces being invoked

Figure 2: The Feature-Interface Graph of JEdit in the software repositories.

4. Empirical Study

In our previous work, we have evaluated the effectiveness of our approach [16]. While in this paper, we improve our previous work aiming at decreasing the time cost and improving the recommendation accuracy. So we define the following research questions to compare with our previous work.

RQ1. Can our approach improve the efficiency of recommending interfaces over our previous work?

RQ2. Can our approach improve the accuracy of the recommended interfaces over our previous work?

4.1. Setup. To evaluate the effectiveness of our approach, we conduct an empirical study on a famous open source software repository, that is, SourceForge (https://sourceforge.net/), to recommend interfaces.

In our approach, we need to use LDA to generate the topics of the projects and interfaces (to save time, we only select 200 active open source projects in the SourceForge repositories). For LDA computation, we used *MALLET* (http://mallet.cs.umass.edu/), which is a highly scalable Java implementation of the *Gibbs* sampling algorithm. We ran for 10,000 sampling iterations, the first 1000 of which were used for parameter optimization. During this process, we need to set the number of topics (K) for LDA analysis. In our previous work, we have conducted some empirical studies to show which number of topics is good for interface recommendation, and the results show that four number of topics is a relatively good choice. So in this study, we also set the number of topics $K = 4$.

In addition, to conduct our study, we invited 20 people to participate in our evaluation. These 20 participants are from all walks of life, such as developers and students. Half of them are from school with 2-3 years of development experience and the other half are from industry with 3-4 years of development experience. In our study, the task for them is to write a text-editing software as in our previous work because all of them used text-editing software frequently and they have the experience of developing this kind of software. These 20 participants are divided into two groups, and they used the previous approach [16] and the new approach in this paper to recommend interfaces, respectively. They performed the study under a similar circumstance.

For *RQ*1, we record the time of the interface recommendation approach in this paper and that in our previous work, respectively.

Table 3: The average time (minutes) and gain (over previous approach) to recommend the interfaces.

Previous approach	New approach	Gain
55.1	26.5	51.9%

For *RQ*2, we ask the participants to provide the feature request for interface recommendation. Then, they evaluate the usefulness of these interfaces to new projects. They evaluate this by the *support* of the interfaces recommended for them, which represents the usefulness of these interfaces that can be used in their development process. The highest score is 4 and the lowest score is 0. The higher the score is, the more useful the interface is. For each participant, top twenty interfaces will be recommended for them. In addition, after their rating on the recommended interfaces, we asked them to discuss on the recommended results and get a consensus on which interfaces would be finally used to develop the new project. This final results on the potentially-be-used interfaces will be used as an authoritative result to measure our previous approach and new approach. To quantitatively compare these two approaches, we used precision and recall, two widely used information retrieval and classification metrics [25], to validate the accuracy of different approaches. Precision measures the fraction of interfaces identified by an interface recommendation approach that are truly relevant (based on the authoritative results), while recall measures the fraction of relevant results (i.e., interfaces that appear in the authoritative results).

4.2. Results. In this subsection, we analyze and discuss the results collected from our studies.

4.2.1. RQ1. In our previous work, Feature-Interface Graph (FIG) for each project in software repositories needs to be generated. While in this paper, we have a step to extract the relevant projects from software repositories and then generate FIGs for these relevant projects. However, during this process, we need to run the LDA topic model to extract the relevant software projects. Moreover, in this paper, we also need to use the LDA topic model to cluster the interfaces for interface generation. In practical software development, efficiency is an important factor for interface reuse. So the first research question is to evaluate whether the approach in this paper can improve the efficiency of our previous work.

Figure 3 shows the time of these twenty participants to get the interfaces recommended by the previous approach and new approach. We notice that the new approach always needs less time to recommend the interfaces. Table 3 shows the average time for these two approaches to generate the interfaces and the gain of new approach over previous approach. From the results, we see that, on average, the time for the new approach to generate the interfaces is 26.5 minutes and 55.1 minutes for previous approach. Moreover, the gain of the new approach over previous approach is bigger than 50%, which is a relatively big scale. Hence, the results indicate that the new approach can save much time to recommend the interfaces, which is more efficient for developers to use. So

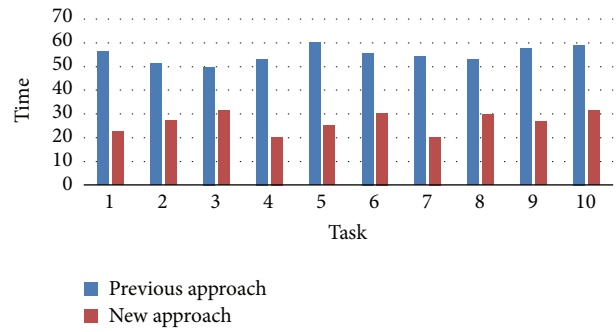


Figure 3: The time to recommend the interfaces for each participant (minutes).

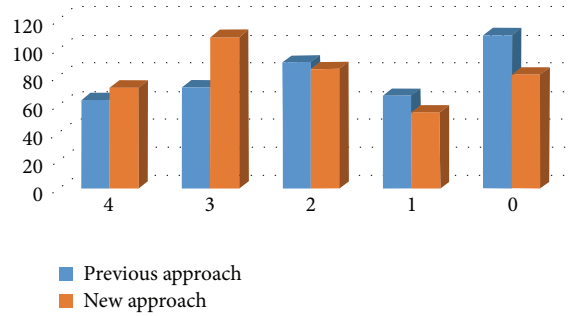


Figure 4: The support results assessed by participants (4 represents more useful, 3 useful, 2 undetermined, 1 useless, and so on).

based on the time results, we can conclude that our approach improves the efficiency over the previous approach.

4.2.2. RQ2. In the new approach, we use a different metric to rank the recommended interfaces. For previous approach, we consider the usage of interfaces in each individual project, while in the new approach, we consider the usage of interfaces from a general view on similar interfaces of the relevant projects. In our previous work, we have shown the accuracy of the recommended interfaces [16], while in the new approach, we use a different metric to recommend the interfaces. So the second research question aims to see whether the recommended interfaces of the new approach are more accurate than the previous approach.

In our study, we select the top twenty interfaces recommended by each approach for participants to assess. So for each participant, he/she needed to evaluate whether each of the twenty interfaces is useful for reuse. So there are in all 400 assessment results. Figure 4 shows the evaluation results of all the participants. From the results, we see that the number of interfaces recommended by the new approach assessed with the highest score (which indicates that the interfaces are more useful) is bigger than that of the previous approach. In addition, the number of interfaces recommended by the new approach assessed with the score (3) is also bigger than that of the previous approach. On the contrary, for interfaces with other scores, that is, 2, 1, and 0, the number of previous approach is bigger than the new approach. In addition, we use the precision and recall to quantitatively compare

Table 4: The precision and recall of these two approaches to recommend the interfaces.

Approach	Precision	Recall
Previous approach	21.2%	44.8%
New approach	27.5%	62.4%
Gain	29.7%	39.3%

the previous approach and new approach. The results are shown in Table 4, which show that both the precision and recall of the new approach are improved. Specifically, the gain approach of the new approach over previous approach is 29.7%, and the gain of recall is 39.3%. So from the results, we notice that the new approach can recommend more useful interfaces for participants to reuse. Hence, we can conclude that the new approach can recommend more accurate interfaces for reuse over the previous approach.

4.3. Threats to Validity. In our study, we have demonstrated the effectiveness of our new approach over the previous approach. But there are still some problems in our evaluation, which threaten the effectiveness of our study.

In our study, the first threat to validity comes from the participants and the project used in the evaluation. The participants in our study engaged in a similar job, but they write their own feature request for interface recommendation. Hence, they may hold different views about text-editing software like *JEdit*. Some of them do not have enough programming experience and do not understand the function of interfaces. In our evaluation, we list the interfaces and functional description of them. Participants need to read these description first and then score for the importance of these interfaces based on their own understanding. We address this threat by using popular and simple words to describe the function of the interfaces and illustrate these descriptions by giving examples. Moreover, these participants performed the study under a similar circumstance, but in practice, we cannot control that they can be fully devoted to the experiment during the empirical study.

In addition, we only choose a text-editing software for participants to conduct our study. We validate the effectiveness of our approach based on the commonly used text-editing software, which is easy and widely used for conducting studies. However, the results cannot be generalized to other projects. Hence, more studies are needed to evaluate our approach on different projects with different characteristics.

The other threat to our validity includes the repositories we used and a possible mismatching between the features provide by the developers and the topics extracted from the open source projects. In addition, the number of topics is set as 4 based on our previous work [16]. But a different value for the number of topics may generate different results.

5. Related Work

A number of approaches have been studied to alleviate the difficulty of code reuse, and some code/software recommendation techniques were proposed, which include the recommendations of example code [26–29], libraries [14] and APIs [30, 31], and software applications [32].

A lot of current related work in this area focused on code or code example recommendation [33–35]. Bruch et al. proposed a code completion system that recommends method calls by looking for code snippets in existing code repositories [36]. They developed this system based on the frequency of method call usage. Proksch et al. also focused on code completion and used Bayesian networks for more intelligent code completion [37]. Lozano et al. proposed a source code recommendation tool that aids developers in software maintenance. They used a genetics-inspired metaphor to analyze source code entities related to the current working context and provided its developer with a number of recommended properties such as naming conventions, used types, and invoked messages [33]. Murakami and Masuhara proposed a method that mechanically evaluates usefulness for their recommendation system called *Selene*. They adjusted several search and user-interface parameters in *Selene* for better recommendation [35]. Then, they also proposed using the user's editing activity to identify the source code relevant to the current method/class. They used a modified degree-of-interest model and incorporated the model in repository-based code recommendation system [28]. McMillan et al. proposed an approach for rapid prototyping that facilitates software reuse by mining feature descriptions and source code from open source repositories. They recommended features and associated source code modules that are relevant to the software product under development [34]. Kim et al. proposed a code example recommendation system that combines the strength of browsing documents and searching for code examples and returns high-quality code example summaries mined from the Web [27]. Ghafari et al. proposed an approach for code recommendation in which code examples are automatically obtained by mining and manipulating unit tests [26]. Zagalsky et al. developed a code search and recommendation tool which brings together social media and code recommendation systems. The tool enables crowd-sourced software development by utilizing both textual and social information [29]. Zhang et al. proposed an approach that recommends interface parameters for code completion. They used an abstract usage instance representation for each API usage example and built a parameter usage database. Then, they queried the database for abstract usage instances in similar contexts and generated parameter candidates by concretizing the instances adaptively [31]. All the above work focus on fine-code level recommendation, which can be used in code completion. In this paper, we recommend higher-level code, that is, interfaces, for developers' reuse, which is different from the above work.

There are also some work that are similar to us, which can recommend libraries or interfaces. Thung et al. proposed two techniques that recommend libraries and APIs [14, 15]. In [14], they proposed a technique that automatically recommended libraries for a particular project. Their input is a set of libraries that the project has used, and the output is the other libraries that are potentially useful for it. Our approach differs in several respects. The input of our approach is the feature of the new project. Then, our approach returns

the potentially useful APIs to the developers based on the matching results between features of the new project and the topics of existing projects. In addition, they also proposed a technique that automatically recommended APIs based on a feature request [15]. They take as input a new textual description and recommended API methods from a set of libraries. They compared the textual description of feature request with the description of the APIs in the API document. Then, they computed the final similarity score between two feature requests and recommended the best suitable APIs. Our work differs from theirs. That is, we studied the relationship between the number of APIs being invoked and the importance of them. Moreover, our approach only needed feature request as the input. Chan et al. proposed a recommendation of API methods through giving textual phrase [13]. Their approach returned a connected APIs undirected graph. Based on this graph and the input query, they mined a subgraph with a particular objective function. In our previous approach [16], we built a graph different from theirs. We recommend the interfaces based on the topic model by matching the feature request and the topics of existing projects. In this paper, we improve our previous work considering further improving the efficiency and accuracy.

There are also a few of work that focused on higher-level application recommendation. McMillan et al. proposed an approach for automatically detecting closely related applications to help users detect similar applications for a given Java application. They used an algorithm that computes a similarity index between Java applications using the notion of semantic layers that correspond to packages and class hierarchies [32]. In addition, they proposed to use API calls from third-party libraries for automatic categorization of software applications that use these API calls [8]. It enables different categorization algorithms to be applied to repositories that contain both source code and bytecode of applications. In this paper, we focused on interface recommendation, and recommendation of applications will become our future concern.

6. Conclusion

In this paper, we improve our previous work [16] to recommend interfaces by using LDA topic model to establish the FIG. We mainly consider two aspects to improve the previous interface recommendation technique. One is that we only identify the relevant open source projects to construct the Feature-Interface Graph to improve the efficiency of the recommendation process. The other is that the interfaces are recommended based on a general usage of the recommended interfaces in all of the relevant software projects in software repositories to improve the recommendation accuracy. The empirical results show that the improved approach in this paper can be more efficient to recommend more accurate interfaces for reuse over the previous work.

In our interface recommendation approach, we still have some issues that remain unsolved. In future, we plan to conduct more studies with some tools that are used to mine API usage and software search such as MAOP [38, 39], Apatite [40], and code quality [41]. We also plan to conduct empirical studies with a large number of other open source projects to show the generality of our approach. Finally, we would like to focus on higher-level recommendation, such as service or application recommendation [42], which considers the recommended interfaces in this paper.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is supported partially by Natural Science Foundation of China under Grants no. 61402396, no. 61472344, and no. 61472343, partially by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant no. 13KJB520027, partially by the Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University under Grant no. KFKT2016B21, partially by the Jiangsu Qin Lan Project, partially by the China Postdoctoral Science Foundation under Grant no. 2015M571489, and partially by the Nantong Application Research Plan under Grant no. BK2014056.

References

[1] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: Finding relevant functions and their usage," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*, pp. 111–120, Honolulu, Hawaii, USA, May 2011.

[2] C. McMillan, D. Poshyvanyk, M. Grechanik, Q. Xie, and C. Fu, "Portfolio: searching for relevant functions and their usages in millions of lines of code," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 4, article 37, 2013.

[3] E. Moritz, M. L. Vasquez, D. Poshyvanyk, M. Grechanik, C. McMillan, and M. Gethers, "ExPort: detecting and visualizing API usages in large source code repositories," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE '13)*, pp. 646–651, Silicon Valley, Calif, USA, November 2013.

[4] S. Bajracharya, J. Ossher, and C. Lopes, "Sourcerer: an infrastructure for large-scale collection and analysis of open-source code," *Science of Computer Programming*, vol. 79, pp. 241–259, 2014.

[5] M. Grechanik, C. McMillan, L. DeFerrari et al., "An empirical investigation into a large-scale Java open source code repository," in *Proceedings of the 4th International Symposium on Empirical Software Engineering and Measurement (ESEM '10)*, Bolzano, Italy, September 2010.

[6] C. McMillan, M. Grechanik, D. Poshyvanyk, C. Fu, and Q. Xie, "Exemplar: a source code search engine for finding highly relevant applications," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1069–1087, 2012.

[7] E. Moritz, M. Linares-Vasquez, D. Poshyvanyk, M. Grechanik, C. McMillan, and M. Gethers, "ExPort: detecting and visualizing API usages in large source code repositories," in *Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering (ASE '13)*, pp. 646–651, IEEE, Silicon Valley, Calif, USA, November 2013.

[8] M. Linares-Vásquez, C. McMillan, D. Poshyvanyk, and M. Grechanik, "On using machine learning to automatically classify software applications into domain categories," *Empirical Software Engineering*, vol. 19, no. 3, pp. 582–618, 2014.

[9] N. Sawadsky, G. C. Murphy, and R. Jiresal, "Reverb: recommending code-related web pages," in *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*, pp. 812–821, May 2013.

[10] S. Wang, D. Lo, and L. Jiang, "Active code search: incorporating user feedback to improve code search relevance," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*, pp. 677–682, Vasteras, Sweden, September 2014.

[11] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "EnTagRec: an enhanced tag recommendation system for software information sites," in *Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME '14)*, pp. 291–300, Victoria, Canada, September 2014.

[12] B. Zhou, X. Xia, D. Lo, C. Tian, and X. Wang, "Towards more accurate content categorization of API discussions," in *Proceedings of the 22nd International Conference on Program Comprehension (ICPC '14)*, pp. 95–105, ACM, Hyderabad, India, June 2014.

[13] W. Chan, H. Cheng, and D. Lo, "Searching connected API subgraph via text phrases," in *Proceedings of the 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE '12)*, 10 pages, Cary, NC, USA, November 2012.

[14] F. Thung, D. Lo, and J. L. Lawall, "Automated library recommendation," in *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE '13)*, pp. 182–191, Koblenz, Germany, October 2013.

[15] F. Thung, S. Wang, D. Lo, and J. Lawall, "Automatic recommendation of API methods from feature requests," in *Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering (ASE '13)*, pp. 290–300, IEEE, Silicon Valley, Calif, USA, November 2013.

[16] W. Shi, X. Sun, B. Li, Y. Duan, and X. Liu, "Using feature-interface graph for automatic interface recommendation: a case study," in *Proceedings of the International Conference on Advanced Cloud and Big Data (CBD '15)*, pp. 296–303, Yangzhou, China, November 2015.

[17] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. 4-5, pp. 993–1022, 2003.

[18] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1565–1616, 2014.

[19] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using IR methods for labeling source code artifacts: is it worthwhile?" in *Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC '12)*, pp. 193–202, Passau, Germany, June 2012.

[20] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE '12)*, pp. 70–79, ACM, Essen, Germany, September 2012.

[21] X. Sun, X. Liu, B. Li, Y. Duan, H. Yang, and J. Hu, "Exploring topic models in software engineering data analysis: a survey," in *Proceedings of the 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD '16)*, 2016.

[22] S. W. Thomas, A. E. Hassan, and D. Blostein, "Mining unstructured software repositories," in *Evolving Software Systems*, pp. 139–162, Springer, Berlin, Germany, 2014.

[23] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolution using topic models," *Science of Computer Programming*, vol. 80, pp. 457–479, 2014.

[24] X. Sun, X. Liu, J. Hu, and J. Zhu, "Empirical studies on the NLP techniques for source code data preprocessing," in *Proceedings of the 3rd International Workshop on Evidential Assessment of Software Technologies (EAST '14)*, pp. 32–39, ACM, Nanjing, China, May 2014.

[25] C. J. van Rijsbergen, *Information Retrieval*, Butterworths, London, UK, 1979.

[26] M. Ghafari, C. Ghezzi, A. Mocci, and G. Tamburrelli, "Mining unit tests for code recommendation," in *Proceedings of the 22nd International Conference on Program Comprehension (ICPC '14)*, pp. 142–145, Hyderabad, India, June 2014.

[27] J. Kim, S. Lee, S.-W. Hwang, and S. Kim, "Enriching documents with examples: a corpus mining approach," *ACM Transactions on Information Systems*, vol. 31, no. 1, article 1, 2013.

[28] N. Murakami, H. Masuhara, and T. Aotani, "Code recommendation based on a degree-of-interest model," in *Proceedings of the 4th International Workshop on Recommendation Systems for Software Engineering (RSSE '14)*, pp. 28–29, Hyderabad, India, 2014.

[29] A. Zagalsky, O. Barzilay, and A. Yehudai, "Example overflow: using social media for code recommendation," in *Proceedings of the 3rd International Workshop on Recommendation Systems for Software Engineering (RSSE '12)*, pp. 38–42, IEEE, Zurich, Switzerland, June 2012.

[30] C. Lv, W. Jiang, Y. Liu, and S. Hu, "APISynth: a new graph-based API recommender system," in *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*, pp. 596–597, Hyderabad, India, June 2014.

[31] C. Zhang, J. Yang, Y. Zhang et al., "Automatic parameter recommendation for practical API usage," in *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pp. 826–836, June 2012.

[32] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pp. 364–374, Zurich, Switzerland, June 2012.

[33] A. Lozano, A. Kellens, and K. Mens, "Mendel: source code recommendation based on a genetic metaphor," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE '11)*, pp. 384–387, Lawrence, Kan, USA, November 2011.

[34] C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher, "Recommending source code for use in rapid software prototypes," in *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pp. 848–858, Zurich, Switzerland, June 2012.

[35] N. Murakami and H. Masuhara, "Optimizing a search-based code recommendation system," in *Proceedings of the 3rd International Workshop on Recommendation Systems for Software Engineering (RSSE '12)*, pp. 68–72, Zurich, Switzerland, June 2012.

[36] M. Bruch, M. Monperrus, and M. Mezini, "Learning from examples to improve code completion systems," in *Proceedings*

of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 213–222, Amsterdam, The Netherlands, August 2009.

[37] S. Proksch, J. Lerch, and M. Mezini, "Intelligent code completion with bayesian networks," *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 1, article 3, 2015.

[38] T. Xie and J. Pei, "MAPO: mining API usages from open source repositories," in *Proceedings of the International Workshop on Mining Software Repositories (MSR '06)*, pp. 54–57, Shanghai, China, May 2006.

[39] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei, "MAPO: mining and recommending API usage patterns," in *Proceedings of the Object-Oriented Programming, 23rd European Conference (ECOOP '09)*, pp. 318–343, Genoa, Italy, July 2009.

[40] D. S. Eisenberg, J. Stylos, A. Faulring, and B. A. Myers, "Using association metrics to help users navigate API documentation," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '10)*, pp. 23–30, Madrid, Spain, September 2010.

[41] L. Han, M. Qian, X. Xu, C. Fu, and H. Kwisaba, "Malicious code detection model based on behavior association," *Tsinghua Science and Technology*, vol. 19, no. 5, pp. 508–515, 2014.

[42] J. Chen, H. Wang, D. Towey, C. Mao, R. Huang, and Y. Zhan, "Worst-input mutation approach to web services vulnerability testing based on SOAP messages," *Tsinghua Science and Technology*, vol. 19, no. 5, pp. 429–441, 2014.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 7519507, 13 pages
http://dx.doi.org/10.1155/2016/7519507

Research Article

On Elasticity Measurement in Cloud Computing

Wei Ai,[1] Kenli Li,[1] Shenglin Lan,[1] Fan Zhang,[2] Jing Mei,[1] Keqin Li,[1,3] and Rajkumar Buyya[4]

[1]*College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China*
[2]*IBM Massachusetts Lab, 550 King Street, Littleton, MA 01460, USA*
[3]*Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*
[4]*Department of Computing and Information Systems, University of Melbourne, Melbourne, VIC 3010, Australia*

Correspondence should be addressed to Kenli Li; lkl@hnu.edu.cn

Received 21 January 2016; Accepted 8 May 2016

Academic Editor: Florin Pop

Elasticity is the foundation of cloud performance and can be considered as a great advantage and a key benefit of cloud computing. However, there is no clear, concise, and formal definition of elasticity measurement, and thus no effective approach to elasticity quantification has been developed so far. Existing work on elasticity lack of solid and technical way of defining elasticity measurement and definitions of elasticity metrics have not been accurate enough to capture the essence of elasticity measurement. In this paper, we present a new definition of elasticity measurement and propose a quantifying and measuring method using a continuous-time Markov chain (CTMC) model, which is easy to use for precise calculation of elasticity value of a cloud computing platform. Our numerical results demonstrate the basic parameters affecting elasticity as measured by the proposed measurement approach. Furthermore, our simulation and experimental results validate that the proposed measurement approach is not only correct but also robust and is effective in computing and comparing the elasticity of cloud platforms. Our research in this paper makes significant contribution to quantitative measurement of elasticity in cloud computing.

1. Introduction

(1) Motivation. As a subscription-oriented utility, cloud computing has gained growing attention in recent years in both research and industry and is widely considered as a promising way of managing and improving the utilization of data center resources and providing a wide range of computing services [1]. Virtualization is a key enabling technology of cloud computing [2]. System virtualization is able to provide abilities to access software and hardware resources from a virtual space and enables an execution platform to provide several concurrently usable and independent instances of virtual execution entities, often called virtual machines (VMs). A cloud computing platform relies on the virtualization technique to acquire more VMs to deal with workload surges or release VMs to avoid resource overprovisioning. Such a dynamic resource provision and management feature is called elasticity. For instance, when VMs do not use all the provided resources, they can be logically resized and be migrated from a group of active servers to other servers, while the idle servers can be switched to the low-power modes (sleep or hibernate) [3].

Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible [4]. By dynamically optimizing the total amount of acquired resources, elasticity is used for various purposes. From the perspective of service providers, elasticity ensures better use of computing resources and more energy savings [5] and allows multiple users to be served simultaneously. From a user's perspective, elasticity has been used to avoid inadequate provision of resources and degradation of system performance [6] and also achieve cost reduction [7]. Furthermore, elasticity can be used for other purposes, such as increasing the capacity of local resources [8, 9]. Hence, elasticity is the foundation of cloud performance and can be considered as a great advantage and a key benefit of cloud computing.

Elastic mechanisms have been explored recently by researchers from academia and commercial fields, and

tremendous efforts have been invested to enable cloud systems to behave in an elastic manner. However, there is no common and precise formula to calculate the elasticity value. Existing definitions of elasticity in the current research literature are all vague concepts and fail to capture the essence of elastic resource provisioning. These formulas of elasticity are not suitable for quantifying and measuring elasticity. Moreover, there is no systematic approach that has been proposed to quantify elastic behavior. Only quantitative elasticity value can produce better comparison between different cloud platforms. Therefore, the measurement of cloud elasticity should be further investigated. As far as we know, the current reported works are ineffective to cover all aspects of cloud elasticity evaluation and measurement. Therefore, we are motivated to develop a comprehensive model and an analytical method to measure cloud elasticity.

(2) Our Contributions. In this paper, we propose a clear and concise definition to compute elasticity value. In order to do that, an elasticity computing model is established by using a continuous-time Markov chain (CTMC). The proposed computing model can quantify, measure, and compare the elasticity of cloud platforms.

The major contributions of this paper are summarized as follows.

(i) First, we propose a new definition of elasticity in the context of virtual machine provisioning and a precise computational formula of elasticity value.

(ii) Second, we develop a technique of quantifying and measuring elasticity by using a continuous-time Markov chain (CTMC) model. We investigate the elastic calculation model intensively and completely. The model is not only an analytical method, but also an easy way to calculate the elasticity value of a cloud platform quantitatively.

(iii) Third, we examine and evaluate our proposed method through numerical data, simulations, and experiments. The numerical data demonstrate the basic parameters which affect elasticity in our analytical model. The simulation results validate the correctness of the proposed method. The experimental results on a real cloud computing platform further show the robustness of our model and method in predicting and computing cloud elasticity.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 describes the definition of cloud elasticity. Section 4 develops the computing model of cloud elasticity. Sections 5, 6, and 7 present simulation and numerical and experimental results, respectively. Section 8 concludes this paper.

2. Related Work

2.1. Elasticity Definition and Measurement. There has been some work on elasticity measurement of cloud computing. In [4], elasticity is described as the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible. In [10], elasticity is defined as the ability of customers to quickly request, receive, and later release as many resources as needed. In [11], elasticity is measured as the ability of a cloud to map a single user's request to different resources. In [12], elasticity is defined as dynamic variation in the use of computer resources to meet a varying workload. In [13], an elastic cloud application or process has three elasticity dimensions, that is, cost, quality, and resources, enabling it to increase and decrease its cost, quality, or available resources, as to accommodate specific requirements. Recently, in [14], elasticity is defined by using the expression $1/(\theta \times \mu)$, where θ denotes the average time to switch from an underprovisioning state to an elevated state and μ denotes the offset between the actual scaling and the autoscaling. Existing definitions of elasticity fail to capture the essence when elastic resource provisioning is performed with virtual machines, and the formulas of elasticity are not suitable for quantifying elasticity. For example, μ in the above expression is difficult to obtain when resource of a cloud is increasing or decreasing. In contrast, the definition proposed in our work reflects the essence of elasticity, and the calculation formula focuses on how to measure the elasticity value effectively.

There are many approaches to predicting elasticity, anticipating the system load behavior, and deciding when and how to scale in/out resources by using heuristics and mathematical/analytical techniques. In [4], the authors established an elasticity metric aiming to capture the key elasticity characteristics. In [15], the authors proposed *execution platforms* and *reconfiguration points* to reflect the proposed elasticity definition. In [5, 7, 16–18], the authors adopted predictive techniques to scale resources automatically. Although these techniques perform well in elasticity prediction, further measurement of elasticity is not covered. In [4], the authors just outlined an elasticity benchmarking approach focusing on special requirements on workload design and implementation. In [15], the authors used thread pools as a kind of elastic resource of the Java virtual machine and presented preliminary results of running a novel elasticity benchmark which reveals the elastic behavior of the thread pool resource. These studies mainly present initial research. In most elasticity work, different elasticity benchmark programs are expected to execute on different systems over varying data sizes and reflect their potential elasticity, but they can only get a macroscopic view of elasticity analysis rather than the calculation of the elasticity value. In contrast, our work performs in-depth research focusing on the measurement of elasticity value.

2.2. Analytical Modeling. Continuous-time Markov chain (CTMC) models have been used for modeling various random phenomena occurring in queuing theory, genetics, demography, epidemiology, and competing populations [19]. CTMC has been applied in a lot of studies to adjust resource allocation in cloud computing. Khazaei et al. proposed an analytical performance model that addresses the complexity of cloud data centers by distinct stochastic submodels using

CTMC [20]. Ghosh et al. proposed a performance model that quantifies power performance trade-offs by interacting stochastic submodels approach using CTMC [21]. Pacheco-Sanchez et al. proposed an analytical performance model that predicts the performance of servers deployed in the cloud by using CTMC [22]. Ghosh et al. proposed a stochastic reward net that quantifies the resiliency of IaaS cloud by using CTMC [23, 24]. However, to the best of our knowledge, CTMC has never been applied in the research of cloud elasticity. Our work in this paper adopts a CTMC model for effective elasticity measurement.

3. Definition of Cloud Elasticity

In this section, we first present a detailed discussion of different states which characterize the elastic behavior of a system. Then, we formally define elasticity that is applied in cloud platforms.

3.1. Notations and Preliminaries. For clarity and convenience, Notations describes the correlated variables which are used in the following sections. To elaborate the essence of cloud elasticity, we give the various states that are used in our discussion. Let i denote the number of VMs in service and let j be the number of requests in the system.

(1) *Just-in-Need State*. A cloud platform is in a just-in-need state if $i < j \leqslant 3i$. T_j is defined as the accumulated time in all just-in-need states.

(2) *Overprovisioning State*. A cloud platform is in an overprovisioning state if $0 \leqslant j \leqslant i$. T_o is defined as the accumulated time in all overprovisioning states.

(3) *Underprovisioning State*. A cloud platform is in an underprovisioning state if $j > 3i$. T_u is defined as the accumulated time in all underprovisioning states.

Notice that constants 1 and 3 in this paper are only for illustration purpose and can be any other values, depending on how an elastic cloud platform is managed. Different cloud users and/or applications may prefer different bounds of the hypothetical just-in-need states. The length of the interval between the upper (e.g., $3i$) and lower (e.g., i) bounds controls the reprovisioning frequency. Narrowing down the interval leads to higher reprovision frequency for a fluctuating workload.

The just-in-need computing resource denotes a balanced state, in which the workload can be properly handled and quality of service (QoS) can be satisfactorily guaranteed. Computing resource overprovisioning, though QoS can be achieved, leads to extra but unnecessary cost to rent the cloud resources. Computing resource underprovisioning, on the other hand, delays the processing of workload and may be at the risk of breaking QoS commitment.

3.2. Elasticity Definition in Cloud Computing. In this section, we present our elasticity definition for a realistic cloud platform and present mathematical foundation for elasticity evaluation. The definition of elasticity is given from a computational point of view and we develop a calculation formula for measuring elasticity value in virtualized clouds. Let T_m be the measuring time, which includes all the periods in the just-in-need, overprovisioning, and underprovisioning states; that is, $T_m = T_j + T_o + T_u$.

Definition 1. The elasticity E of a cloud perform is the percentage of time when the platform is in just-in-need states; that is, $E = T_j/T_m = 1 - T_o/T_m - T_u/T_m$.

Broadly defining, elasticity is the capability of delivering preconfigured and just-in-need virtual machines adaptively in a cloud platform upon the fluctuation of the computing resources required. Practically it is determined by the time needed from an underprovisioning or overprovisioning state to a balanced resource provisioning state. Definition 1 provides a mathematical definition which is easily and accurately measurable. Cloud platforms with high elasticity exhibit high adaptivity, implying that they switch from an overprovisioning or an underprovisioning state to a balanced state almost in real time. Other cloud platforms take longer time to adjust and reconfigure computing resources. Although it is recognized that high elasticity can also be achieved via physical host standby, we argue that, with virtualization-enabled computing resource provisioning, elasticity can be delivered in a much easier way due to the flexibility of service migration and image template generation.

Elasticity E reflects the degree to which a cloud platform changes upon the fluctuation of workloads and can be measured by the time of resource scaling by the quantity and types of virtual machine instances. We use the following equation to calculate its value:

$$E = 1 - \frac{(T_o + T_u)}{T_m} = 1 - \frac{T_o}{T_m} - \frac{T_u}{T_m}, \quad (1)$$

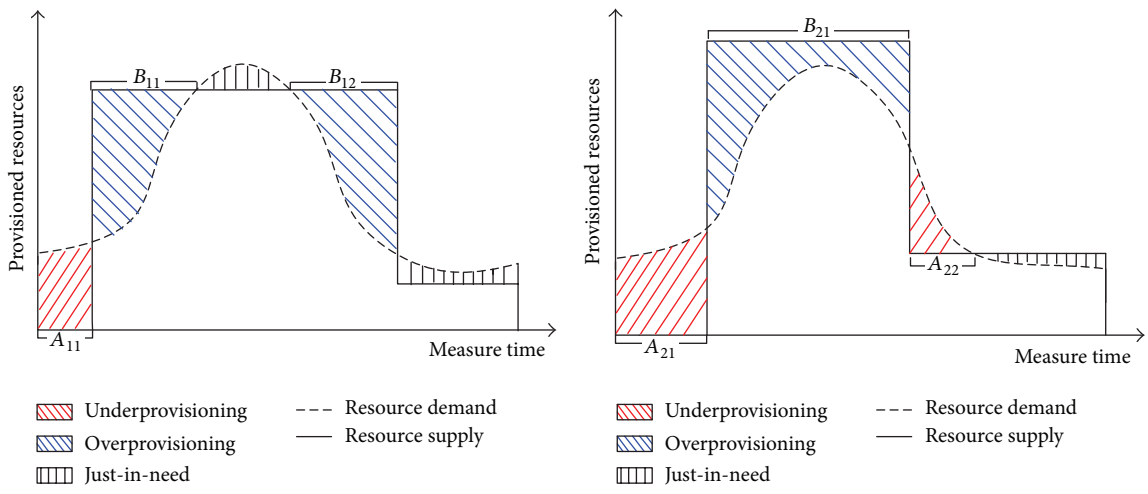
where T_m denotes the total measuring time, in which T_o is the overprovisioning time which accumulates each single period of time that the cloud platform needs to switch from an overprovisioning state to a balanced state and T_u is the underprovisioning time which accumulates each single period of time that the cloud platform needs to switch from an underprovisioning state to a corresponding balanced state.

Let P_j, P_o, and P_u be the accumulated probabilities of just-in-need states, overprovisioning states, and underprovisioning states, respectively. If T_m is sufficiently long, we have $P_j = T_j/T_m$, $P_o = T_o/T_m$, and $P_u = T_u/T_m$. Therefore, we get

$$E = P_j = 1 - P_o - P_u. \quad (2)$$

Equation (1) can be used when elasticity is measured by monitoring a real system. Equation (2) can be used when elasticity is calculated by using our CTMC model. If elasticity metrics are well defined, elasticity of cloud platforms could easily be captured, evaluated, and compared.

We would like to mention that the primary factors of elasticity, that is, the amount, frequency, and time of resource reprovisioning, are all summarized in T_o and T_u (i.e., P_o and P_u). Elasticity can be increased by changing these factors. For example, one can maintain a list of standby or underutilized compute nodes. These nodes are prepared for the upcoming



(a) Elastic cloud resource provisioning in cloud platform A

(b) Elastic cloud resource provisioning in cloud platform B

Figure 1: An example of elasticity metrics.

surge of workload, if there is any, to minimize the time needed to start these nodes. Such a hot standby strategy increases cloud elasticity by reducing T_u.

3.3. An Example. In Figure 1, $A_{11} = 3$ hours, $A_{21} = 5$ hours, and $A_{22} = 4$ hours are the time spans in underprovisioning states, and $B_{11} = 4$ hours, $B_{12} = 5$ hours, and $B_{21} = 10$ hours are the time spans in overprovisioning states. The measuring time of cloud platform A is $T_m^A = 24$ hours and cloud platform B is $T_m^B = 26$ hours. So $T_u^A = A_{11} = 3$ hours (i.e., underprovisioning time of cloud platform A), $T_o^A = B_{11} + B_{12} = 9$ hours (i.e., overprovisioning time of cloud platform A), $T_u^B = A_{21} + A_{22} = 9$ hours (i.e., underprovisioning time of cloud platform B), and $T_o^B = B_{21} = 10$ hours (i.e., overprovisioning time of cloud platform B). According to (1), the elasticity value of cloud platform A is $E^A = 1 - T_o^A/T_m^A - T_u^A/T_m^A = 0.5$, and the elasticity value of cloud platform B is $E^B = 1 - T_o^B/T_m^B - T_u^B/T_m^B = 0.27$. As can be seen, a greater elasticity value would exhibit better elasticity.

3.4. Relevant Properties of Clouds. In this section, we compare cloud elasticity with a few other relevant concepts, such as cloud resiliency, scalability, and efficiency.

Resiliency. Laprie [25] defined resiliency as the persistence of service delivery that can be trusted justifiably, when facing changes. Therefore, cloud resiliency implies (1) the extent to which a cloud system withstands the external workload variation and under which no computing resource reprovisioning is needed and (2) the ability to reprovision a cloud system in a timely manner. We think the latter implication defines the cloud elasticity while the former implication only exists in cloud resiliency. In our elasticity study, we will focus on the latter one.

Scalability. Elasticity is often confused with scalability in more ways than one. Scalability reflects the performance speedup when cloud resources are reprovisioned. In other words, scalability characterizes how *well* in terms of performance a new compute cluster, either larger or smaller, handles a given workload. On the other hand, elasticity explains how *fast* in terms of the reprovisioning time the compute cluster can be ready to process the workload. Cloud scalability is impacted by quite a few factors such as the compute node type and count and workload type and count. For example, Hadoop MapReduce applications typically scale much better than other single-thread applications. It can be defined in terms of scaling number of threads, processes, nodes, and even data centers. Cloud elasticity, on the other hand, is only constrained by the capability that a cloud service provider offers. Other factors that are relevant to cloud elasticity include the type and count of standby machines, computing resources that need to be reprovisioned. Different from cloud scalability, cloud elasticity does not concern workload/application type and count at all.

Efficiency. Efficiency characterizes how cloud resource can be efficiently utilized as it scales up or down. This concept is derived from speedup, a term that defines a relative performance after computing resource has been reconfigured. Elasticity is closely related to efficiency of the clouds. Efficiency is defined as the percentage of maximum performance (speedup or utilization) achievable. High cloud elasticity results in higher efficiency. However, this implication is not always true, as efficiency can be influenced by other factors independent of the system elasticity mechanisms (e.g., different implementations of the same operation). Scalability is affected by cloud efficiency. Thus, efficiency may enhance elasticity, but not sufficiency. This is due to the fact that elasticity depends on the resource types, but efficiency is not limited by resource types. For instance, with a multitenant architecture, users may exceed their resources quota. They may compete for resources or interfere each other's job executions.

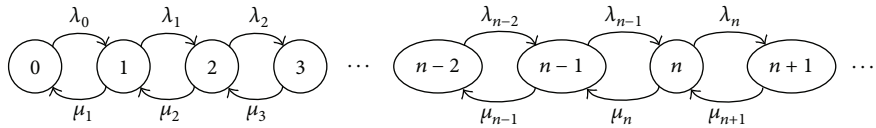


Figure 2: State-transition-rate diagram for a birth-death process.

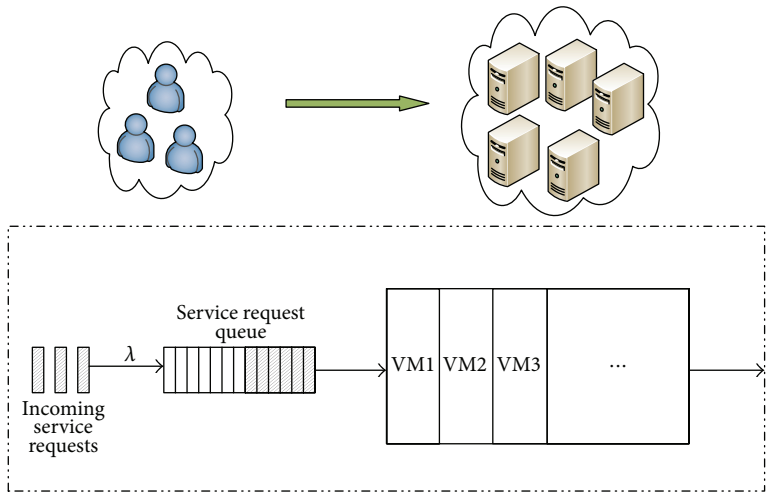


Figure 3: Modeling an elastic cloud computing platform as an extended $M/M/m$ queuing system.

4. Elasticity Analysis Using CTMC

In this paper, we implement the cloud elasticity computing model using CTMC.

4.1. A Queuing Model. This section mainly explains why the continuous-time Markov chain (CTMC) can be applied to compute cloud elasticity and the connection between them.

A continuous-time Markov chain is a continuous time, discrete-state Markov process. Many CTMC have transitions that only go to neighboring states, that is, either up one or down one; they are called birth-and-death processes. Motivated by population models, a transition up one is called a birth, while a transition down one is called a death. The birth rate in state i is denoted by λ_i, while the death rate in state i is denoted by μ_i. The state-transition-rate diagram for a birth-and-death process (with state space $\{0, 1, \ldots, n\}$) takes the simple linear form shown in Figure 2.

In many applications, it is natural to use birth-and-death processes. One of the queuing models is $M/M/m$ queue, which has m servers and unlimited waiting room. The main properties of a queuing system are as follows.

(1) Requests arrive in a Poisson process with parameter λ.

(2) The service times are exponential random variables with parameter μ.

So a queuing system is a birth-and-death process with Markov property.

A cloud computing service provider serves customers' service requests by using a multiserver system. An elastic cloud computing platform treated as a multiserver system and modeled as an extended $M/M/m$ queuing system is shown in Figure 3. Assume that service requests arrive by following a Poisson process and task service times are independent and identically distributed random variables that follow an exponential distribution. When a running request finishes, the capacity used by the corresponding VM is released and becomes available for serving the next request. The request at the head of the queue is processed (i.e., first-come-first-served) on a running VM if there is capacity to run a scheduled request. Elastic resource provisioning cannot be done with physical machines, and only virtual machines can be reconfigured in real time. A cloud platform is able to adapt to variation in workload by starting up or shutting off VMs in an autonomic manner, avoiding overprovisioning or underprovisioning. If no enough running VMs are available (e.g., underprovisioning state), a new VM is started up and used for service. If there are excessive VMs (e.g., overprovisioning state), redundant VMs are shut off.

According to (1) and (2), the calculation of the elasticity value needs to count the accumulated time in all the overprovisioning and underprovisioning states. In real cloud platforms, it is possible to record the overprovisioning time and underprovisioning times. Furthermore and fortunately, the accumulated probability of both overprovisioning and underprovisioning states can be computed using our proposed CTMC model as discussed in the next section.

4.2. Elastic Cloud Platform Modeling. To model elastic cloud platforms, we make the following assumptions.

(i) All VMs are homogeneous with the same service capability and are added/removed one at a time.

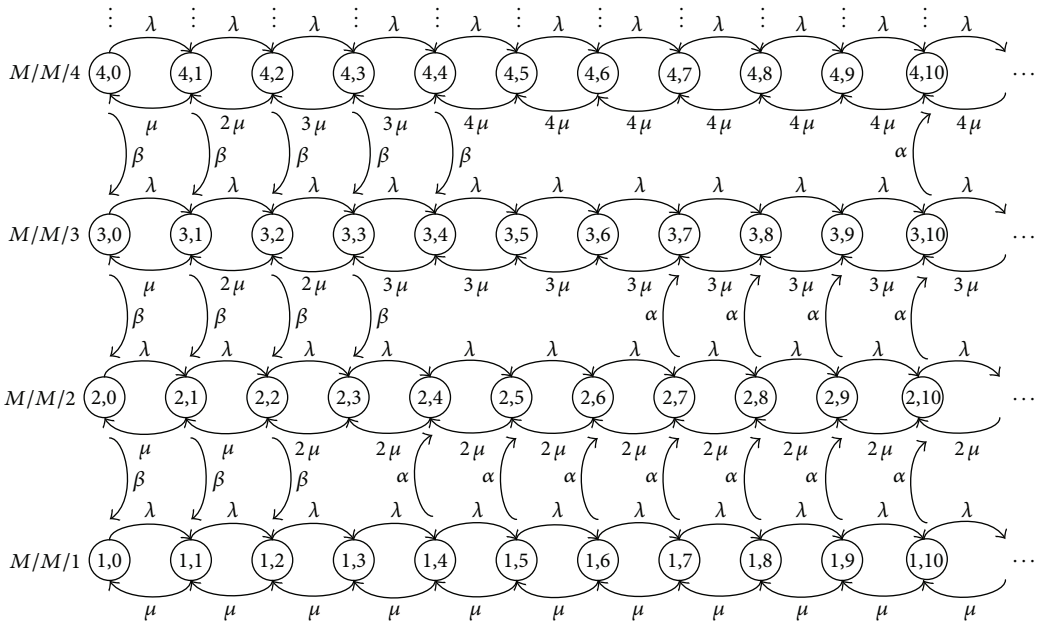


Figure 4: State-transition-rate diagram of our extended $M/M/m$ queuing system.

(ii) The user request arrivals are modeled as a Poisson process with rate λ.

(iii) The service time, the start-up time, and the shut-off time of each VM are governed by exponential distributions with rates μ, α, and β, respectively [26].

(iv) Let i denote the number of virtual machines that are currently in service, and let j denote the number of requests that are receiving service or in waiting.

(v) Let statev(i, j) denote the various states of a cloud platform when the virtual machine number is i and the request number is j. Let the hypothetical just-in-need state, overprovisioning state, and underprovisioning state be JIN, OP, and UP, respectively. We can set the equations of the relation between the virtual machine number and the request number as follows:

$$\text{statev}(i, j) = \begin{cases} \text{OP}, & \text{if } 0 \leq j \leq i; \\ \text{JIN}, & \text{if } i < j \leq 3i; \\ \text{UP}, & \text{if } j > 3i. \end{cases} \quad (3)$$

The hypothetical just-in-need state, overprovisioning state, and underprovisioning state are listed in Table 1.

Based on these assumptions, we build a two-dimensional continuous-time Markov chain (CTMC) for our extended $M/M/m$ queuing system shown in Figure 4, which is actually a mixture of $M/M/m$ systems for all $m = 1, 2, 3, \ldots$. The CTMC model records the number of VMs and the number of user requests received for service, which can eventually be employed to calculate the elastic value E.

Each state in the model, shown in Figure 4, is labeled as (i, j), where i ($i \in \{1, \ldots, m\}$) denotes the number of virtual machines that are currently processing requests and

Table 1: The relation between the virtual machine number and the request number.

VM number	Overprovisioning state	Just-in-need state	Underprovisioning state
1	$0 \leqslant j \leqslant 1$	$1 < j \leqslant 3$	$j > 3$
2	$0 \leqslant j \leqslant 2$	$2 < j \leqslant 6$	$j > 6$
3	$0 \leqslant j \leqslant 3$	$3 < j \leqslant 9$	$j > 9$
4	$0 \leqslant j \leqslant 4$	$4 < j \leqslant 12$	$j > 12$
⋮	⋮	⋮	⋮
i	$0 \leqslant j \leqslant i$	$i < j \leqslant 3i$	$j > 3i$
⋮	⋮	⋮	⋮

j ($j \in \{0, 1, \ldots, m\}$) denotes the number of requests that are receiving service. For the purpose of numerical calculation, we set the maximum number of VMs that can be deployed as m, which is sufficiently large to guarantee enough accuracy. Similarly, the maximum j is m. Let μ be the service rate of each VM. So the total service rate for each state is the product of number of running VMs and μ.

The state transition in an elastic cloud computing model can occur due to user request arrival, service completion, virtual machine start-up, or virtual machine shut-off. In state (i, j), according to Table 1, the state can be determined as "just-in-need," "underprovisioning," or "overprovisioning." Depending on the upcoming event, four possible transitions can occur.

Case 1. When a new request arrives, the system transits to state $(i, j + 1)$ with rate λ.

Case 2. When a requested service is completed, if the system examines the state as not "overprovisioning," the system

moves back to state $(i, j-1)$ with total service rate $i\mu$. If the system examines the state as "overprovisioning" and $i = j$, the system moves back to state $(i, j-1)$ with total service rate $(j-1)\mu$, because a server is shutting off and cannot perform any task at the moment. If the system examines the state as "overprovisioning" and $i \neq j$, the system moves back to state $(i, j-1)$ with total service rate $j\mu$.

Case 3. The system examines the state as "underprovisioning" and transits to state $(i+1, j)$ with rate α.

Case 4. The system examines the state as "overprovisioning" and transits to state $(i-1, j)$ with rate β.

We use $P_{i,j}$ to denote the steady-state probability that the system stays in state (i, j), where $i \in \{1, \ldots, m\}$ and $j \in \{0, 1, \ldots, m\}$. We can now set the balance equations as follows:

$$
\begin{aligned}
\lambda P_{i,j} &= K_2 \mu P_{i,j+1} + \beta P_{i+1,j}, && \text{if } i = 1,\ j = 0;\\
(\lambda + K_1\mu) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1} + \beta P_{i+1,j}, && \text{if } i = 1,\ 0 < j \le i+1;\\
(\lambda + K_1\mu) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1}, && \text{if } i = 1,\ i+1 < j \le 3i;\\
(\lambda + K_1\mu + \alpha) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1}, && \text{if } i = 1,\ 3i < j < m;\\
(K_1\mu + \alpha) P_{i,j} &= \lambda P_{i,j-1}, && \text{if } i = 1,\ j = m;\\
(\lambda + \beta) P_{i,j} &= K_2 \mu P_{i,j+1} + \beta P_{i+1,j}, && \text{if } 1 < i < m,\ j = 0;\\
(\lambda + K_1\mu + \beta) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1} + \beta P_{i+1,j}, && \text{if } 1 < i < m,\ 0 < j \le i;\\
(\lambda + K_1\mu) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1} + \beta P_{i+1,j}, && \text{if } 1 < i < m,\ j = i+1;\\
(\lambda + K_1\mu) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1}, && \text{if } 1 < i \le m,\ i+1 < j \le 3(i-1);\\
(\lambda + K_1\mu) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1} + \alpha P_{i-1,j}, && \text{if } 1 < i < m,\ 3(i-1) < j \le 3i;\\
(\lambda + K_1\mu + \alpha) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1} + \alpha P_{i-1,j}, && \text{if } 1 < i < m,\ 3i < j < m;\\
(K_1\mu + \alpha) P_{i,j} &= \lambda P_{i,j-1} + \alpha P_{i-1,j}, && \text{if } 1 < i < m,\ j = m;\\
(\lambda + \beta) P_{i,j} &= K_2 \mu P_{i,j+1}, && \text{if } i = m,\ j = 0;\\
(\lambda + K_1\mu + \beta) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1}, && \text{if } i = m,\ 0 < j \le i+1;\\
(\lambda + K_1\mu) P_{i,j} &= \lambda P_{i,j-1} + K_2 \mu P_{i,j+1} + \alpha P_{i-1,j}, && \text{if } i = m,\ 3(i-1) < j < m;\\
K_1 \mu P_{i,j} &= \lambda P_{i,j-1} + \alpha P_{i-1,j}, && \text{if } i = m,\ j = m,
\end{aligned}
\tag{4}
$$

where

$$
\begin{aligned}
&K_1 = j, \quad \text{if } i > j;\\
&K_1 = j-1, \quad \text{if } i = j,\ j \neq 1;\\
&K_1 = 1, \quad \text{if } i = j,\ j = 1;\\
&K_1 = i, \quad \text{if } i < j;\\
&K_2 = j+1, \quad \text{if } i > j+1;\\
&K_2 = j, \quad \text{if } i = j+1,\ j+1 \neq 1;\\
&K_2 = 1, \quad \text{if } i = j+1,\ j+1 = 1;\\
&K_2 = i, \quad \text{if } i < j+1,
\end{aligned}
\tag{5}
$$

$$
\sum_{i=1}^{m} \sum_{j=0}^{m+1} P_{i,j} = 1.
$$

In the above equations, λ, μ, α, and β are the request arrival rate (i.e., the interarrival times of service requests are independent and identically distributed exponential random variables with mean $1/\lambda$), the service rate (i.e., the average number of tasks that can be finished by a VM in one unit of time), the virtual machine start-up rate (i.e., a VM needs time $T = 1/\alpha$ to turn on), and the virtual machine shut-off rate (i.e., a VM needs time $T = 1/\beta$ to shut down), respectively. The balance equations link the probabilities of entering and leaving a state in equilibrium. The total number of equations is $m \times (m+1) + 1$, but there are only $m \times (m+1)$ variables: $P_{1,0}, P_{1,1}, \ldots, P_{m,m}$. Therefore, in order to derive $P_{i,j}$, we need to remove one of the equations to obtain the unique equilibrium solution. Unfortunately, the steady-state balance equations cannot be solved in a closed form; hence, we must resort to a numerical solution.

The input and output parameters of our CTMC model are summarized in the following.

Input. The request arrival rate is λ, the service rate is μ, the virtual machine start-up rate is α, and the virtual machine shut-off rate is β. (In addition, the definitions of "just-in-need," "underprovisioning," and "overprovisioning" states should also be included.)

Output

(i) The accumulated underprovisioning state probability P_u of a cloud platform is as follows:

$$P_u = \sum_{i=1}^{m} \sum_{j=3i+1}^{m+1} P_{i,j}, \quad (6)$$

where $P_{i,j}$ is the steady-state probability.

(ii) The accumulated overprovisioning state probability P_o of a cloud platform is as follows:

$$P_o = \sum_{i=2}^{m} \sum_{j=0}^{i} P_{i,j}, \quad (7)$$

where $P_{i,j}$ is the steady-state probability.

(iii) The elasticity value E of a cloud platform is obtained by (2), (6), and (7).

5. Model Analysis

In this section, we present some numerical results obtained based on the proposed elastic cloud platform modeling, illustrating and quantifying the elasticity value under different load conditions and different system parameters. All the numerical data in this section are obtained by setting m = 1,000, that is, the maximum number of VMs that can be deployed, to guarantee sufficient numerical accuracy.

5.1. Varying the Arrival Rate. For the first scenario, we have considered a system with different service rates (μ = 100, 120, 140, 160, and 180 jobs/hour), while the arrival rate is a variable from λ = 100 to 400 jobs/hour in sixteen steps. In all cases, the virtual machine start-up rate and virtual machine shut-off rate are assigned values of α = 120 VMs/hour and β = 540 VMs/hour.

Figure 5 illustrates that the elasticity value is an increasing function of the arrival rate. As can been seen, it increases rather quickly when the arrival rate is up to 300 and smoothly when the arrival rate is higher. This behavior is due to the fact that increasing λ results in noticeable reduction of the probability of overprovisioning but slight change of the probability of underprovisioning. Furthermore, it is observed that the elasticity value decreases as the service rate increases, as described in the next section.

5.2. Varying the Service Rate. For the second scenario, we have considered a system with different arrival rates (λ = 200, 220, 240, 260, and 280 jobs/hour), while the service rate is a variable from μ = 10 to 290 jobs/hour in fifteen steps. In all cases, the virtual machine start-up rate and virtual machine shut-off rate are assigned values of α = 120 VMs/hour and β = 540 VMs/hour.

Figure 6 illustrates that the elasticity value is a decreasing function of the service rate. It shows that, for a fixed arrival rate, increasing service rate decreases the elasticity value sharply and almost linearly. This phenomenon is due to the fact that increasing μ results in noticeable increment of the probability of overprovisioning, and change of the probability of underprovisioning does not affect the decreasing trend of the just-in-need probability. Figure 6 also confirms that the elasticity value is an increasing function of the arrival rate.

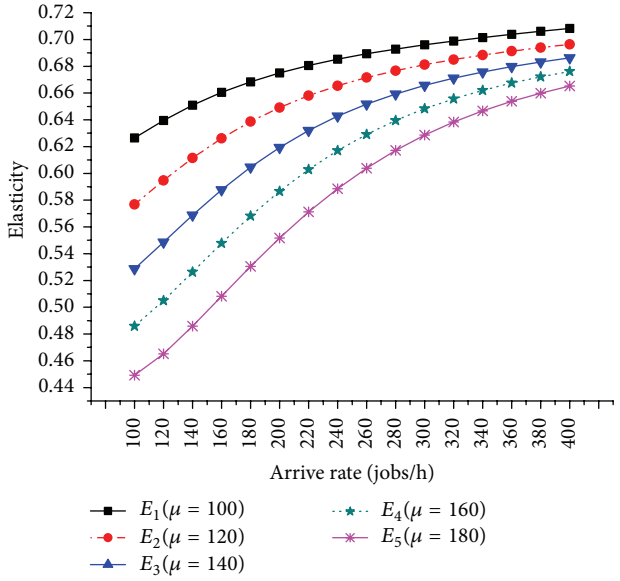


Figure 5: Elasticity versus arrival rate.

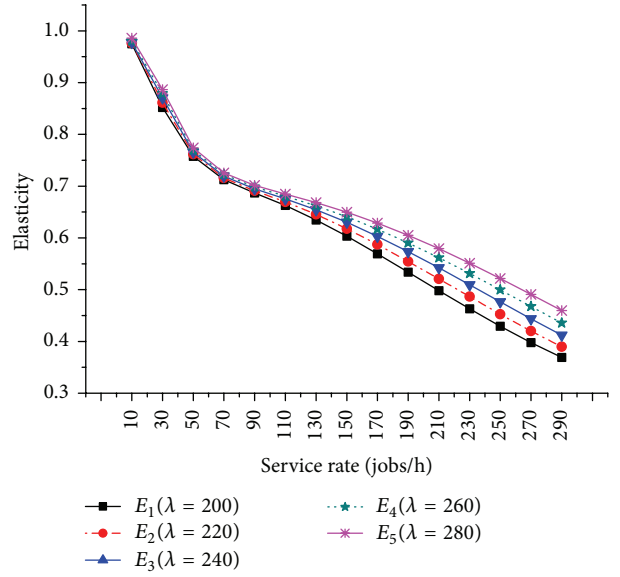
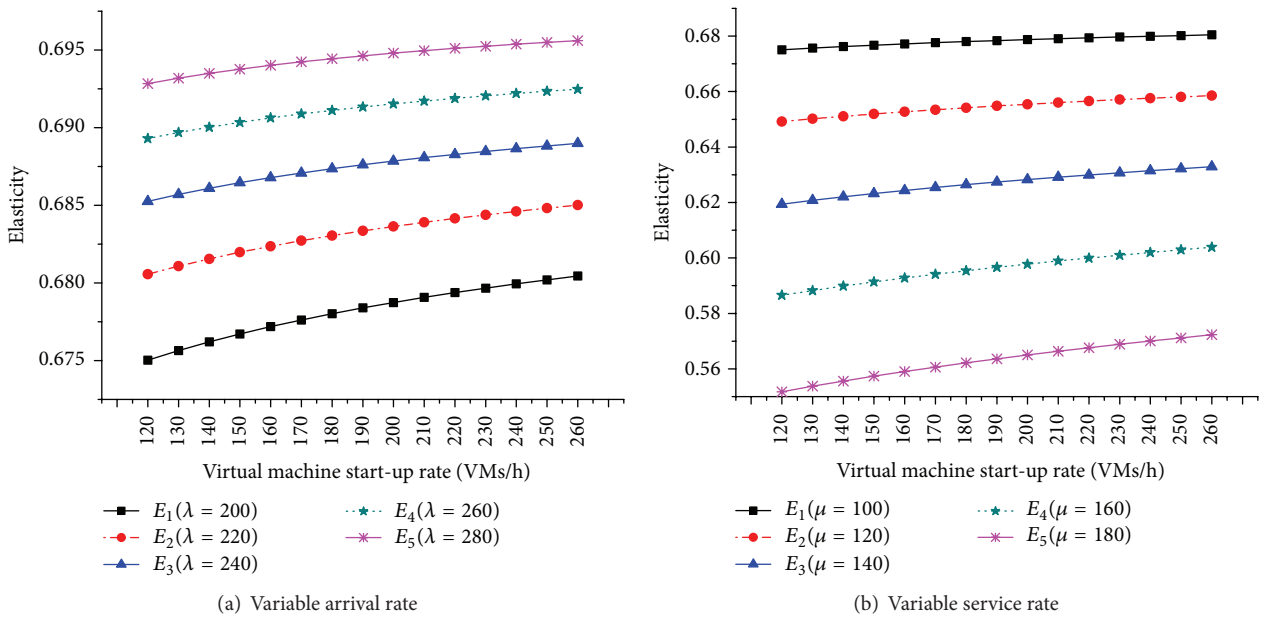


Figure 6: Elasticity versus service rate.

5.3. Varying the Virtual Machine Start-Up Rate. For the third scenario, Figure 7 shows numerical results for a fixed arrival rate, service rate, and virtual machine shut-off rate but different virtual machine start-up rates.

First, we characterize the elasticity value by presenting the effect of different arrival rates (λ = 200, 220, 240, 260, and 280 jobs/hour) and the virtual machine start-up rate is a variable from α = 120 to 260 VMs/hour in fifteen steps. In all cases, other system parameters are set as follows. The service rate is



(a) Variable arrival rate (b) Variable service rate

FIGURE 7: Elasticity versus virtual machine start-up rate.

$\mu = 100$ jobs/hour, and the virtual machine shut-off rate is $\beta = 540$ VMs/hour. As can be seen in Figure 7(a), it increases slightly when the virtual machine start-up rate increases. This is due to the fact that, for high virtual machine start-up rate, each virtual machine start-up time is shorter, meaning that less time is needed to switch from an underprovisioning state to a balanced state, so the probability of underprovisioning P_u is smaller, while the probability of overprovisioning P_o does not change too much, and the probability of just-in-need P_j is increasing.

Second, we also analyze the effects of different service rates ($\mu = 100$, 120, 140, 160, and 180 jobs/hour), while the virtual machine start-up rate is a variable from $\alpha = 120$ to 260 VMs/hour in fifteen steps. In all cases, other system parameters are set as follows. The arrival rate is $\lambda = 200$ jobs/hour, and the virtual machine shut-off rate is $\beta = 540$ VMs/hour. It can be seen in Figure 7(b) that the elasticity value increases slightly with increasing virtual machine start-up rate.

The results allow us to conclude the increasing elasticity value at increasing virtual machine start-up rate for a fixed arrive rate, service rate, and virtual machine shut-off rate. In other words, increasing the virtual machine start-up rate will decrease the probability of underprovisioning and increase the just-in-need probability. These behaviors (see Figure 7) confirm that the elasticity value of a cloud platform has a relationship to its virtual machine start-up speed.

5.4. Varying the Virtual Machine Shut-Off Rate. For the fourth scenario, Figure 8 shows numerical results for a fixed arrival rate, service rate, and virtual machine start-up rate but different virtual machine shut-off rates.

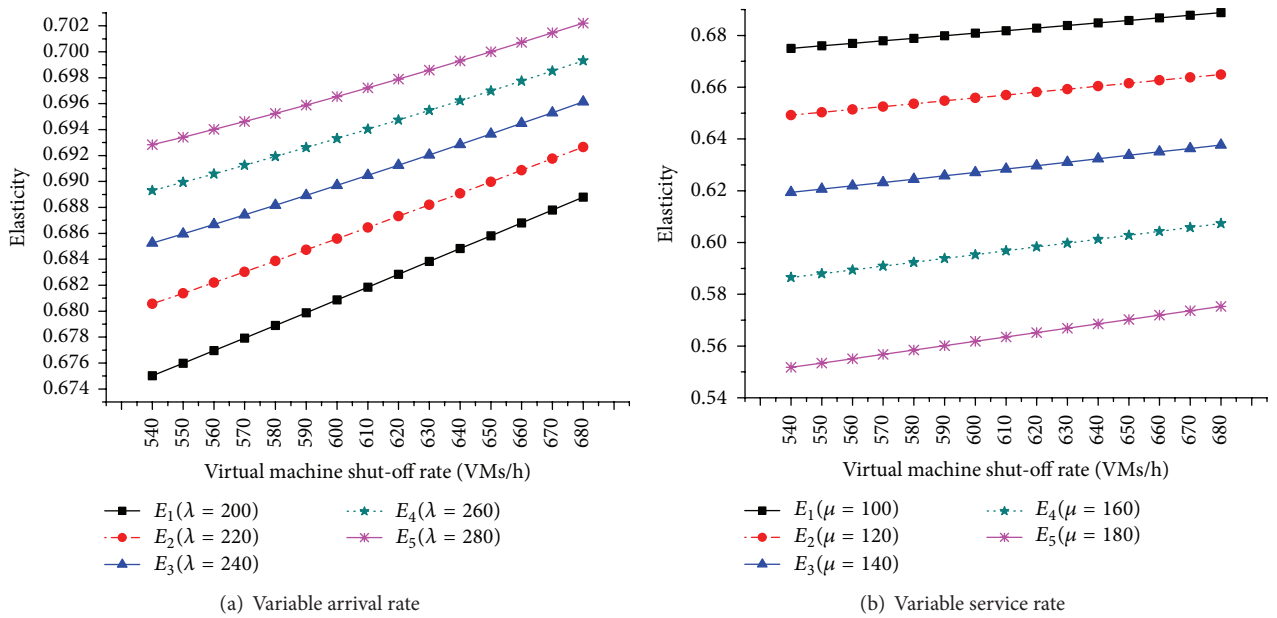
We examine the effect of virtual machine shut-off rate on elasticity. For different arrival rates ($\lambda = 200$, 220, 240, 260, and 280 jobs/hour), the virtual machine shut-off rate is a variable from $\beta = 540$ to 680 VMs/hour in fifteen steps. In all cases, other system parameters are set as follows. The service rate is $\mu = 100$ jobs/hour, and the virtual machine start-up rate is $\alpha = 120$ VMs/hour. It can be seen from Figure 8(a) that the elasticity value increases slightly where the virtual machine shut-off rate is increased from 540 to 680 VMs/hour. This happens because the virtual machine shut-off time is shorter, and a platform becomes more responsive, resulting in diminishing overprovisioning time which is the accumulate time for the system to switch from an overprovisioning state to a balanced state. Furthermore, the probability of overprovisioning P_o is smaller, the probability of underprovisioning P_u shows slight change, and the probability of just-in-need P_j is increasing.

We also calculate the elasticity value under the different service rates ($\mu = 100, 120, 140, 160$, and 180 jobs/hour), while the virtual machine shut-off rate is a variable from $\beta = 540$ to 680 VMs/hour in fifteen steps. The arrival rate is $\lambda = 200$ jobs/hour, and the virtual machine start-up rate is $\alpha = 120$ VMs/hour. In Figure 8(b), the elasticity value increases slightly by increasing the virtual machine shut-off rate. This is also because of the corresponding reduction in virtual machine shut-off time, guaranteeing shorter overprovisioning probability P_o.

Based on these results, we can conclude the increasing elasticity value at increasing virtual machine shut-off rate for a fixed arrive rate, service rate, and virtual machine start-up rate. In other words, increasing the virtual machine shut-off rate will decrease the probability of overprovisioning and increase the just-in-need probability. These behaviors of Figure 8 confirm that the elasticity value of a cloud platform has a relationship to its virtual machine shut-off speed.

6. Simulation Results

In this section, we present our elastic cloud simulation system called *Cloud Elasticity Value*. Its aim is to demonstrate



(a) Variable arrival rate (b) Variable service rate

FIGURE 8: Elasticity versus virtual machine shut-off rate.

that our elasticity measurement is correct and effective in computing and comparing the elasticity value and to show cloud elasticity under different parameter settings.

6.1. Design of the Simulator. Our simulation uses the same code base for the elasticity measurement as the real implementation. The simulator is implemented in about 40,000 lines of C++ code. It runs in a Linux box over a rack-mount server with Intel®Core™2 Duo CPU and 4.00 GB of memory.

The simulator consists of four modules, that is, the task generator module, the virtual machine monitor module, the request monitor module, and the queue module. The task generator module produces simulation of Poisson distribution requests. The virtual machine monitor module is used for deciding whether to start up and shut off the virtual machines and recording the start-up and shut-off times. The request monitor module is used to count how many requests are being serviced in the system and to record the service times. Arrived service requests are first placed in a queue module and recorded their arrival times before they are processed by any virtual machine.

The main process of the simulator is listed as follows.

Step 1. The task generator module produces simulation of Poisson distribution requests. When the service requests arrive, they are placed in a queue module and recorded their arrival times.

Step 2. The virtual machine monitor module determines whether to start up or shut off the VMs and records the start-up and shut-off times.

Step 3. The request monitor module determines whether there is a request in the queue. If there is a request, it takes a request and assigns it to a running virtual machine and records the service time.

Step 4. After the simulation time is over, the simulator counts up the accumulated time in overprovisioning and underprovisioning states and returns the elasticity value using (1).

6.2. Simulation Results and Analysis. We have evaluated cloud elasticity values using two methods, that is, (1) the elasticity values in terms of the steady-state probabilities obtained for the given parameter and (2) the elasticity values in terms of our simulation system obtained for the same parameters. We compare our CTMC model solutions with the results produced by the simulation method.

We have considered the arrival rate characterized by λ = 60, 200, and 600 jobs/hour. The service rate values chosen are μ = 60, 200, and 600 jobs/hour. In all cases, the virtual machine start-up rate is assigned the value of α = 300 VMs/hour, while the virtual machine shut-off rate is β = 540 VMs/hour.

Table 2 shows the difference between the elasticity values obtained by the CTMC model and the simulator. From Table 2, we can see that the elasticity values between the two cases are very close, with the maximum relative difference only 0.8 percent. The agreement between the simulation and CTMC model results is excellent, which confirms the validity of our CTMC model. So we conclude that the proposed elasticity quantifying and measuring method using the continuous-time Markov chain (CTMC) model is correct and effective.

7. Experiments on Real Systems

7.1. Experiment Environment. We have conducted our experiments on LuCloud, a cloud computing environment located in Hunan University. On top of hardware and Ubuntu Linux 12.04 operating system, we install KVM virtualization

Table 2: Comparison of CTMC model results and simulation results.

Arrival rate	Service rate	Start-up rate	Shut-off rate	Method		Difference
				CTMC model	Simulation	
60	60	300	540	0.705212	0.702837	0.3%
60	200	300	540	0.445756	0.475257	0.4%
60	600	300	540	0.635151	0.637240	0.3%
200	60	300	540	0.735167	0.739055	0.5%
200	200	300	540	0.543948	0.546414	0.5%
200	600	300	540	0.235727	0.234727	0.4%
600	60	300	540	0.827098	0.828784	0.2%
600	200	300	540	0.688974	0.684784	0.6%
600	600	300	540	0.435171	0.438784	0.8%

Table 3: Comparison of CTMC model results and experimental results with exponential service times.

Arrival rate	Service rate	Start-up rate	Shut-off rate	Method		Difference
				CTMC model	Experiment	
60	60	120	540	0.701205	0.706082	0.6%
60	200	120	540	0.475480	0.479752	0.9%
60	600	120	540	0.631525	0.649275	3.0%
200	60	120	540	0.731117	0.739533	1.2%
200	200	120	540	0.516099	0.521308	1.0%
200	600	120	540	0.221065	0.269039	2.2%
600	60	120	540	0.817088	0.826455	1.1%
600	200	120	540	0.687533	0.681169	0.9%
600	600	120	540	0.409537	0.491034	0.9%

software which virtualizes the infrastructure and provides unified computing and storage resources. To create a cloud environment, we install CloudStack open-source cloud environment, which is composed of a cluster and responsible for global management, resource scheduling, task distribution, and interaction with users. The cluster is managed by a cloud manager (8 AMD Opteron Processor 4122 CPU, 8 GB memory, and 1 TB hard disk). We use our elasticity testing platform to achieve the allocation of resources, that is, virtual machine start-up and shut-off on LuCloud.

7.2. Experiment Process and Results. First, in order to validate the proposed model, Table 3 summarizes the comparison between the two approaches, that is, the CTMC model and the experiments on LuCloud. We have considered the arrival rate characterized by $\lambda = 60$, 200, and 600 jobs/hour. The service rate values chosen are $\mu = 60$, 200, and 600 jobs/hour. The virtual machine start-up rate is assigned the value of $\alpha = 120$ VMs/hour, and the virtual machine shut-off rate is assigned the value of $\beta = 540$ VMs/hour.

In Table 3, we can observe that the elasticity values of both approaches are very close, with the maximum relative difference only 3.0 percent. We conclude that the proposed CTMC model can be used to compute the elasticity of cloud platforms and can offer accurate results within reasonable difference.

Our second set of experiments focus on the robustness of our model and method, that is, its applicability when the assumptions of our model are not satisfied. We have considered Gamma distributions for the service times. The Gamma(k, θ) distribution is defined in terms of a shape parameter k and a scale parameter θ [27]. We use the same parameter settings in Table 3, except that the exponential distributions of service times are replaced by Gamma distributions, that is, Gamma$(0.0083, 2)$, Gamma$(0.0025, 2)$, and Gamma$(0.00083, 2)$, such that the service rates are still $\mu = 60$, 200, and 600 jobs/hour.

From Table 4, we can see that the elasticity values of the CTMC model and the experiments are very close, with the maximum relative difference only 3.3 percent. We observe that the experimental results with Gamma service times match very closely with those of the proposed CTMC model. So we conclude that the proposed model and method for quantifying and measuring elasticity using continuous-time Markov chain (CTMC) are not only correct and effective, but also robust and applicable to real cloud computing platforms.

8. Conclusion

In this paper, we have introduced a new definition of cloud elasticity. We have presented an analytical method suitable for evaluating the elasticity of cloud platforms, by using a continuous-time Markov chain (CTMC) model. Validation of the analytical results through extensive simulations has shown that our analytical model is sufficiently detailed to capture all realistic aspects of resource allocation process,

Table 4: Comparison of CTMC model results and experimental results with Gamma service times.

Arrival rate	Start-up rate	Shut-off rate	Service distribution	Method		Difference
				CTMC model	Experiment	
60	120	540	Gamma(0.0083, 2)	0.701205	0.716401	2.2%
60	120	540	Gamma(0.0025, 2)	0.475480	0.476752	0.3%
60	120	540	Gamma(0.00083, 2)	0.631525	0.612930	3.0%
200	120	540	Gamma(0.0083, 2)	0.731117	0.711420	2.7%
200	120	540	Gamma(0.0025, 2)	0.516099	0.522762	1.3%
200	120	540	Gamma(0.00083, 2)	0.221065	0.227146	2.8%
600	120	540	Gamma(0.0083, 2)	0.817088	0.835865	2.3%
600	120	540	Gamma(0.0025, 2)	0.687533	0.667146	3.0%
600	120	540	Gamma(0.00083, 2)	0.409537	0.395865	3.3%

that is, virtual machine start-up and virtual machine shut-off, while maintaining excellent accuracy between CTMC model results and simulation results. We have examined the effects of various parameters including request arrival rate, service time, virtual machine start-up rate, and virtual machine shut-off rate. Our experimental results further evidence that the proposed measurement approach can be used to compute cloud elasticity in real cloud platforms. Consequently, cloud providers and users can obtain quantitative, informative, and reliable estimation of elasticity, based on a few essential characterizations of a cloud computing platform.

Notations

E: The elasticity value
i: The number of VMs in service
j: The number of requests in the queue
T_j: The accumulated just-in-need time
T_o: The accumulated overprovisioning time
T_u: The accumulated underprovisioning time
T_m: The measuring time
P_j: The accumulated probability of just-in-need states
P_o: The accumulated probability of overprovisioning states
P_u: The accumulated probability of underprovisioning states
λ: The request arrival rate
μ: The request service rate
α: The virtual machine start-up rate
β: The virtual machine shut-off rate
(i, j): A state in our CTMC model
$P_{i,j}$: The steady-state probability of state (i, j)
N: The average number of requests in the queue
T: The average response time
M: The average number of VMs in service
CPR: The cost-performance ratio.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grants nos. 61133005 and 61432005) and the National Natural Science Foundation of China (Grants nos. 61370095 and 61472124).

References

[1] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 45–58, 2014.

[2] M. Bourguiba, K. Haddadou, I. E. Korbi, and G. Pujolle, "Improving network I/O virtualization for cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 673–681, 2014.

[3] Cost-efficient consolidating service for Aliyun's cloud-scale computing, http://kylinx.com/papers/c4.pdf.

[4] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: what it is, and what it is not," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC '13)*, pp. 23–27, San Jose, Calif, USA, June 2013.

[5] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, p. 5, ACM, Cascais, Portugal, October 2011.

[6] G. Galante and L. C. E. de Bona, "A survey on cloud computing elasticity," in *Proceedings of the IEEE/ACM 5th International Conference on Utility and Cloud Computing (UCC '12)*, pp. 263–270, Chicago, Ill, USA, November 2012.

[7] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS '11)*, pp. 559–570, IEEE, Minneapolis, Minn, USA, July 2011.

[8] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid clouds," *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861–870, 2012.

[9] J. O. Fitó, Í. Goiri, and J. Guitart, "SLA-driven elastic cloud hosting provider," in *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP '10)*, pp. 111–118, IEEE, Pisa, Italy, February 2010.

[10] L. Badger, T. Grance, R. Patt-Corner, and J. Voas, *Draft Cloud Computing Synopsis and Recommendations*, vol. 800, NIST Special Publication, 2011.

[11] R. Cohen, *Defining Elastic Computing*, 2009, http://www.elasticvapor.com/2009/09/defining-elastic-computing.html.

[12] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud Computing: Principles and Paradigms*, vol. 87, John Wiley & Sons, New York, NY, USA, 2010.

[13] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong, "Principles of elastic processes," *IEEE Internet Computing*, vol. 15, no. 5, pp. 66–71, 2011.

[14] K. Hwang, X. Bai, Y. Shi, M. Li, W. Chen, and Y. Wu, "Cloud performance modeling with benchmark evaluation of elastic scaling strategies," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 130–143, 2016.

[15] M. Kuperberg, N. Herbst, J. von Kistowski, and R. Reussner, *Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms*, KIT, Fakultät für Informatik, 2011.

[16] W. Dawoud, I. Takouna, and C. Meinel, "Elastic VM for cloud resources provisioning optimization," in *Advances in Computing and Communications*, A. Abraham, J. L. Mauri, J. F. Buford, J. Suzuki, and S. M. Thampi, Eds., vol. 190 of *Communications in Computer and Information Science*, pp. 431–445, Springer, Berlin, Germany, 2011.

[17] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Proceedings of the IEEE 4th International Conference on Cloud Computing (CLOUD '11)*, pp. 500–507, IEEE, Washington, DC, USA, July 2011.

[18] Z. Gong, X. Gu, and J. Wilkes, "Press: predictive elastic resource scaling for cloud systems," in *Proceedings of the International Conference on Network and Service Management (CNSM '10)*, pp. 9–16, IEEE, Ontario, Canada, October 2010.

[19] W. J. Anderson, *Continuous-Time Markov Chains*, Springer Series in Statistics: Probability and Its Applications, Springer, New York, NY, USA, 1991.

[20] H. Khazaei, J. Mišić, V. B. Mišić, and S. Rashwand, "Analysis of a pool management scheme for cloud computing centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 5, pp. 849–861, 2013.

[21] R. Ghosh, V. K. Naik, and K. S. Trivedi, "Power-performance trade-offs in IaaS cloud: a scalable analytic approach," in *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W '11)*, pp. 152–157, IEEE, Hong Kong, June 2011.

[22] S. Pacheco-Sanchez, G. Casale, B. Scotney, S. McClean, G. Parr, and S. Dawson, "Markovian workload characterization for QoS prediction in the cloud," in *Proceedings of the IEEE 4th International Conference on Cloud Computing (CLOUD '11)*, pp. 147–154, IEEE, Washington, Wash, USA, July 2011.

[23] R. Ghosh, F. Longo, V. K. Naikz, and K. S. Trivedi, "Quantifying resiliency of IaaS cloud," in *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems*, pp. 343–347, IEEE, New Delhi, India, November 2010.

[24] R. Ghosh, D. Kim, and K. S. Trivedi, "System resiliency quantification using non-state-space and state-space analytic models," *Reliability Engineering & System Safety*, vol. 116, pp. 109–125, 2013.

[25] J.-C. Laprie, "From dependability to resilience," in *Proceedings of the 38th IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. G8–G9, Anchorage, Alaska, USA, June 2008.

[26] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proceedings of the IEEE 5th International Conference on Cloud Computing (CLOUD '12)*, pp. 423–430, IEEE, Honolulu, Hawaii, USA, June 2012.

[27] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Task execution time modeling for heterogeneous computing systems," in *Proceedings of the IEEE 9th Heterogeneous Computing Workshop (HCW '00)*, pp. 185–199, Cancun, Mexico, 2000.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 9315493, 11 pages
http://dx.doi.org/10.1155/2016/9315493

Research Article

Optimizing Checkpoint Restart with Data Deduplication

Zhengyu Chen, Jianhua Sun, and Hao Chen

College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China

Correspondence should be addressed to Jianhua Sun; jhsun@hnu.edu.cn

Received 1 March 2016; Accepted 5 May 2016

Academic Editor: Laurence T. Yang

The increasing scale, such as the size and complexity, of computer systems brings more frequent occurrences of hardware or software faults; thus fault-tolerant techniques become an essential component in high-performance computing systems. In order to achieve the goal of tolerating runtime faults, checkpoint restart is a typical and widely used method. However, the exploding sizes of checkpoint files that need to be saved to external storage pose a major scalability challenge, necessitating the design of efficient approaches to reducing the amount of checkpointing data. In this paper, we first motivate the need of redundancy elimination with a detailed analysis of checkpoint data from real scenarios. Based on the analysis, we apply inline data deduplication to achieve the objective of reducing checkpoint size. We use DMTCP, an open-source checkpoint restart package, to validate our method. Our experiment shows that, by using our method, single-computer programs can reduce the size of checkpoint file by 20% and distributed programs can reduce the size of checkpoint file by 47%.

1. Introduction

Infrastructure as a Service (IaaS) [1] is a form of cloud computing that provides virtualized computing resources over the Internet. IaaS allows customers to rent computing resources from large data centers rather than buy and maintain dedicated hardware. More and more software production and services are directly deployed and run on IaaS clouds. With the increase of computing nodes, the probability of failures increases. Hardware failure fail-over or any unexpected reason makes node inactive. For distributed and high-performance computing (HPC) applications, a failure in one node causes the whole system to fail.

Fault tolerance is an indispensable component of cloud computing. However, due to the embarrassingly parallel nature of mainstream cloud applications, most approaches are designed to deal with fault tolerance of individual processes and virtual machines. This either involves restarting failed tasks from scratch (e.g., MapReduce) or leveraging live migration to replicate the state of virtual machines on-the-fly in order to be able to switch to a backup virtual machine in case the primary instance has failed [2, 3].

Checkpoint is an important method to provide fault tolerance for distributed and HPC applications [4]. Through the use of checkpoint, program's in-memory states can be written to persistent storage. If a crash occurs, the previously saved checkpoint can be used to recover program to the latest state. In IaaS cloud environments, checkpoint technique can be more beneficial. For example, when resource prices become expensive or the budget is tight, we can suspend the running program and recover it later, or we can transfer the program to a new cloud provider without losing progress.

However, with the increasing scale of computation, the checkpoint size also increases. So the efficiency of the checkpoint operation is becoming more and more important. Not only will checkpoints incur performance cost, but the checkpoint files consume large storage space, which causes I/O bottlenecks and generates extra operational costs. For example, saving 1 GB RAM for each 1,000 processes consumes 1 TB of space. Although there have been attempts to reduce the IO overhead using hardware technology (i.e., SSDs), with the increasing of the number of computational nodes, this method is still not an ideal solution.

In this paper, we focus on the reduction of the checkpoint file size and ultimately achieve the purpose of reducing the I/O and storage overhead. In the storage space of the checkpoint file is mainly consumed by the content of the process's address space. We analyzed the content of the process's address space in order to identify different segments

of redundancy. The results of analysis show that there exist a lot of duplicate data in the stack section. For distributed applications, in heap segment between different processes, there are a lot of duplicate data, and the contents in their dynamic link library and code segments are the same. But the amount of duplicate data in dynamic link library is less. According to these characteristics, we propose a method that uses inline deduplication to identify and eliminate duplicate memory contents at the page level to reduce the checkpoint file size. DMTCP (Distributed Multithreaded Checkpointing) is a transparent user-level checkpoint package for distributed applications. We implement our method based on DMTCP. Experiments show that our method can greatly reduce the checkpoint size.

The contributions of our work are as follows:

(i) We conduct a detailed analysis of the contents of the checkpoint file, which reveals some characteristics of data redundancy and the potential of integrating deduplication into existing checkpoint systems.

(ii) Motivated by our experimental analysis, we propose a method to reduce the size of the checkpoint file, which works for both single-node and distributed high-performance computing systems.

(iii) We present the design and implementation of our approach in DMTCP and perform extensive evaluation on a set of representative applications.

This paper is organized as follows. Section 2 presents related work. Section 3 describes our motivation and related background. Section 4 contains the design and implementation of our approach. In Section 5, we provide an experimental evaluation of our design. Section 6 analyzes the experimental results, and the conclusions and future work are given in Section 7.

2. Related Work

2.1. Checkpoint Restart. There are roughly four major directions of research on reducing the size of checkpoint file. Incremental checkpointing reduces the checkpoint size by saving only the changes made by the application from the last checkpoint [5–7]. Usually, a fixed number of incremental checkpoints is created in between two full ones. During a restart, the state is restored by using the most recent full checkpoint file and applying in an ordered manner all the differences before resuming the execution.

Memory exclusion [8] skips temporary or unused buffers to reduce the size of the checkpoint file size. This is done by providing an interface to the application to specify regions of memory that can be safely excluded from the checkpoint.

Checkpoint compression is a method for the reduction of the checkpoint file size by reducing the size of process images before writing them to stable storage [9]. Besides compression, another possible solution is deduplication that is the mainstream storage technology. It can effectively optimize storage capacity. This technology can reduce the requirement on physical storage space and can meet the data storage need which grows day by day. Our work focuses on inline deduplication to identify and eliminate duplicate data in the process.

2.2. Data Deduplication. Data deduplication is a specialized data compression technique for eliminating duplicate copies of repeating data [10, 11]. Data deduplication splits the input file into a set of data blocks and calculates a fingerprint for each of them. If there exists a block sharing the same fingerprint, it indicates that block is duplicated, and we only need to store the index number for the duplicated block. Otherwise, it means that data block is unique, and its content needs to be stored. As can be seen from the above process, the key technology of deduplication mainly includes data block segmentation, fingerprint computation, and data block retrieval. Among these techniques, data block segmentation is the most crucial.

Block segmentation algorithm is divided into three types: the fixed-size partition, content-defined chunking, and sliding block. Fixed-size partition algorithm first splits the input data into fixed-size chunks. Then, chunks are compared between each other in order to identify and eliminate duplicates. While simple and fast, fixed-size partition algorithm is very sensitive to data insertion and deletion and not adaptive to the changes of content.

To deal with such issues, content-defined approaches [12] are proposed. Essentially, they involve a sliding window over the data and that hashes the window content at each step using Rabin's fingerprinting method [13]. When a particular condition is fulfilled, a new chunk border is introduced and the process is repeated until all input data was processed, leading to a collection of variable-sized chunks. Content-defined chunking algorithm is not sensitive to changes in content, data insertion and deletion only affects a few blocks, and the remaining data blocks are not affected. But it also has disadvantages, the size of the data block is difficult to determine; coarse granularity results in nonoptimal effect, and fine-grained granularity often leads to higher cost. Thus, the most difficult part of this algorithm lies in how to choose a suitable granularity.

Sliding block algorithm [14] combines the advantages of fixed-size partition algorithm and content-defined chunking algorithm. This algorithm uses Rabin fingerprinting to subdivide byte-streams into chunks with a high probability of matching other chunks generated likewise. If a signature of the chunck/block matches one of the precomputed or prespecified delimiters, the algorithm designates the end of this window as a chunk boundary. Once the boundary has been identified, all bytes starting from the previous known chunk boundary to the end of the current window is designated a chunk. A hash of this new chunk is computed and compared against the signatures of all preexisting chunks in the system. In practice, this method makes use of four tuning parameters, namely, the minimum chunk size, the maximum chunk size, the average chunk size, and the window size. The sliding block algorithm deals with data insertion and data deletion process efficiently and can detect more redundant data than content-defined approaches; its drawback is that it is prone to data fragmentation. In order to obtain more redundant information, we use the sliding

block algorithm in our approach, which will be described later.

At present, data deduplication technology is widely used in the storage system and network system; by using deduplication it can effectively reduce the data storage and system overhead. For example, Srinivasan et al. [15] can achieve 60–70% of the maximum deduplication with less than a 5% CPU overhead and a 2–4% latency impact through the use of deduplication. Agarwal et al. [16] designed EndRE; the system uses the technology of data deduplication to eliminate the redundancy of network data and reduce the cost of WAN access. The experiment results show that EndRE can save an average of 26% of the bandwidth and reduce the end-to-end delay of 30%.

3. Background and Motivation

3.1. Background. Checkpoint restart is a mechanism that periodically saves the state of an application to persistent storage and offers the possibility to resume the application from such intermediate states.

Depending on the transparency with regard to the application program, single-process checkpoint techniques can be classified as application-level, user-level, or system-level. User-level checkpoint services are implemented in user space but are transparent to the user application. This is achieved by virtualizing all system calls to the kernel, without being tied to a particular kernel [17]. System-level checkpointing services are either implemented inside the kernel or as a kernel module [18]. The checkpoint images in system-level checkpointing are not portable across different kernels.

DMTCP (Distributed Multithreaded Checkpointing) is a transparent user-level checkpoint package for distributed applications [19]. DMTCP is very convenient for applications to set checkpoint and restart. It works completely in user space and does not require any changes to the application or operating system. DMTCP automatically tracks all local and remote child processes and their relationships. DMTCP implements a coordinator process because DMTCP can also checkpoint distributed computations across many computers. The user can issue a command to the coordinator, which will then relay the command to each of the user processes of the distributed computation.

Like the principles of a checkpoint, DMTCP also copies the program information in-memory to the checkpoint file. Program's in-memory information includes the process id, the process's address space, opened file information, and signal state. In the checkpoint file, the content of the process's address space occupies the main storage space. The process's address space information is mainly read from */proc/self/maps*, from which we can obtain the contents of the address space. The program's address space consists of heap, stack, shared libraries, *mmap* memory area, data, and text segment.

3.2. Motivation. In this paper, we focus on reducing the size of checkpoint file based on deduplication. In this section, we analyze the content redundancy in regular checkpoint files. The content needed to be stored by the checkpoint files generated by DMTCP can be classified into 5 categories: heap, stack, code, dynamic link library, and mmap. We ignore other types of content in checkpoint files due to the limited space they occupy. The target programs used in our experiment can be divided into two types: single-node program and distributed program. Next, we conduct analysis on program address spaces of these five categories on the two kinds of programs, respectively, in order to characterize data duplication in prospective applications. After retrieving the content of their progress address space, three methods can be used to perform the analysis: gzip compression, deduplication, and the hybrid of compression and deduplication. Sliding block algorithm is used in deduplication, the minimum chunk size is 512 B, the maximum chunk size is 32768 (32 K), average chunk size is 4096 B (4 KB), and the window size is 48 B.

Table 1: Single-node program heap (KB).

Type	Original	gzip	Deduplication	Deduplication + gzip
BaseLine	132	0.49	68.18	0.51
Python	868	128.24	600	130.20
Vim	2220	283.14	2107.46	289.68
BC	264	8.8	232.23	8.83
Perl	264	23.63	264	23.98

3.2.1. Single-Node Program Analysis. We perform experiments using the following applications: BaseLine, a simple C program, whose functionality is to print numbers consistently; Python(2.7.6), an interpreted, interactive, object-oriented programming language; Vim(7.4), an advanced text editor; BC(1.06.95), an arbitrary precision calculator language; and Perl(5.18.2), Practical Extraction and Report Language interpreter.

Table 1 displays the experimental results on heap. The compression rate of BaseLine, Vim, BC, and Perl is 99.6%, 87%, 96.6%, and 91%, respectively, showing that gzip based compression has good effect on heap. But the results become unsatisfying when redundant data deletion technique is used. For example, in BaseLine, the original size of heap is 132 KB; after deduplication it still occupies 68.18 KB, with a deduplication rate of 48.3%. For some applications the results are even worse. For example, the rate for Python and Vim is 30.8% and 5.1%, respectively. The results become even worse if we try to use gzip directly succeeding the deduplication procedure. As shown in Table 1, after the deduplication + gzip operations, the resulting heap size is even bigger than when only the gzip compression approach is used. Of course, the redundant data comes from adding unnecessary index information.

Table 2 illustrates that applying deduplication and deduplication + gzip to stack is much more effective as compared to heap. In BaseLine, the original stack size is 8072 KB; after compression it shrinks to 11.69 KB, and after deduplication it is 105.1 KB. With deduplication + gzip, the size is only 4.15 KB, which means the compression rate is 99.86% and the deduplication rate is 98.7%.

In single-node program experiments, we can conclude that the duplication of heap is quite limited as compared to its

Table 2: Single-node program stack (KB).

Type	Original	gzip	Deduplication	Deduplication + gzip
BaseLine	8072	11.69	105.16	4.15
Python	8072	13.19	137.24	5.75
Vim	8068	13.03	101.36	5.57
BC	8072	13.32	105.18	5.8
Perl	8072	12.37	105.28	4.86

Table 3: Distributed program heap (KB).

Type	Original	Deduplication	Redundancy rate
BaseLine	264	101.59	61.52%
CG	264	165.14	37.54%
EP	264	149.07	43.53%
LU	264	153.09	42%
MG	264	161.13	38.98%
IS	264	129	51.13%

relatively high compression efficiency. On the other hand, the compression and duplication rates are both high in the case of stack, and using gzip + deduplication can achieve much better effect. During the analysis of code, dynamic link library, and mmap, we find that the duplication rate is not high, and gzip compression is more suitable for reducing the size of these components.

3.2.2. Distribute Program Analysis. We conduct experiments on distributed programs using the following applications: BaseLine and NAS NPB3.3. The BaseLine is a simple MPI program, whose function is to calculate Pi. The NAS Parallel Benchmarks (NPB) are a small set of programs designed to help evaluate the performance of parallel supercomputers. NPB 3.3-MPI was used in our experiments. The benchmarks run under MPICH2 are CG (Conjugate Gradient, irregular memory access and communication, level C), EP (Embarrassingly Parallel, level C), LU (Lower-Upper Gauss-Seidel Solver, level C), MG (Multigrid on a sequence of meshes, long- and short-distance communication, memory intensive, level C), and IS (Integer Sort, random memory access, level C). For convenience, we adopt two computers to build the experiment cluster. A single process is run on each node. The content of various segments, that is, heap, stack, code, DLL, and mmap of each process, is collected. "Original" in Tables 3 and 4 represents the total size of each segment of the two nodes.

Table 3 shows the results of the heap of distributed programs. The original size of BaseLine is 264 KB. After deduplication, the size becomes 201.59 KB, obtaining a duplication rate of 61.52%. Table 3 also shows that the duplication rates of CG, EP, LU, MG, and IS are 37.54%, 43.53%, 42%, 38.98%, and 51.13%, respectively. The rates increase significantly as compared to the single-program counterparts, inferring that there exists a lot of duplicated information on the heap of each process between the two different nodes. We can apply gzip + deduplication method to reduce the heap size of distributed programs.

Table 4: Distributed program stack (KB).

Type	Original	Deduplication	Redundancy rate
BaseLine	16140	80.21	99.5%
CG	16144	88.24	99.45%
EP	16140	80.24	99.5%
LU	16136	88.27	99.45%
MG	16144	88.27	99.45%
IS	16136	76.2	99.53%

Table 4 displays the result of the stack of distributed programs, which implies that there is a lot of redundant data in stacks. For example, the original size of BaseLine is 16140 KB, and the size becomes 80.21 KB after deduplication, obtaining a duplication rate of 99.5%. Other testing applications also reveal a very high duplication rate. The rates of CG, EP, LU, MG, and IS are 99.45%, 99.5%, 99.45%, 99.45%, and 99.53%, respectively. In conclusion, not only do stacks have a lot of redundant data themselves, but also the duplication rates of the stack between different processes are even higher.

Other segments, like code, DLL, and mmap, do not have too much redundant data on the same node, but their contents are the same between different nodes in distributed programs. But DMTCP stores every code, mmap, and DLL of each node when setting checkpoints. Thus we only need one checkpoint file to store the code and mmap of multiple processes on the same node and DLL, while others only need to keep the index.

In this section, we analyze the duplication rate of heap, stack, DLL, and mmap in single program and distributed program, respectively. In order to reduce the checkpoint file size efficiently, we implement different strategies on different segments, which can be summarized as follows:

Single-node program:

(i) for heap, code, DLL, mmap, and others: we continue to use gzip for compression;
(ii) for stack: gzip + deduplication method is adopted.

Distributed program:

(i) for heap and stack between different processes: gzip + deduplication method is adopted;
(ii) for code, DLL, and mmap of multiple processes on the same node: only one copy is stored in the checkpoint file; others just keep the corresponding index;
(iii) other: we continue to use gzip compression.

4. Design and Implementation

4.1. Overview. In this paper, we focus on how to reduce the checkpoint file size. When setting a checkpoint, we examine each memory page in the program and apply techniques tailored to the page type as detailed in the following.

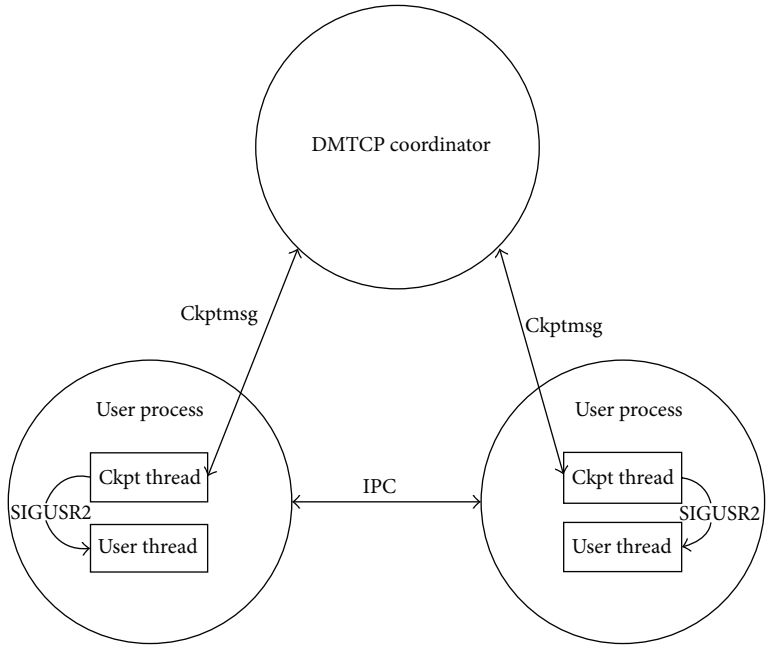


Figure 1: Architecture of DMTCP.

(i) Heap: in single-node programs, redundancy in heap is not high, so gzip compression is used. In distributed programs, we employ a combination of deduplication and gzip to identify and eliminate redundancy within heap pages.

(ii) Stack: for both single-node programs and distributed programs, we employ a combination of deduplication and gzip to identify and eliminate redundancy within stack pages.

(iii) DLL and mmap of multiple processes on the same node and code: in single programs, we only use gzip compression. In distributed programs, for DLL, code, and mmap of multiple processes on the same node, only one copy is stored in the checkpoint file, while a corresponding index is kept.

(iv) Other: for both single-node programs and distributed programs, the remaining pages are compressed using gzip.

Figure 1 shows the architecture of DMTCP. DMTCP uses a stateless centralized process, the coordinator, to synchronize checkpoint and restart between distributed processes. In each user process, DMTCP creates a checkpoint thread. The checkpoint thread is used to receive commands sent from the coordinator, such as setting up checkpoints. In addition, the checkpoint thread contacts user thread through the signal (SIGUSR2). Our approach is implemented based on DMTCP. Next, we discuss in detail the specific implementation.

4.2. Heap and Stack. The application can request a new checkpoint at any time by using the command: *dmtcp_command-c*. When this command is issued and received by the coordinator, each node will start to set checkpoint. From the previous analysis, we can know that heap segment redundancy in single-node program is not high, and stack segments have higher duplicate data. In a distributed application, heap and stack segment between different processes have a high possibility of containing duplication. We need to identify the single-node program and distributed programs. So, first we need to check the type of the program. The process of each node sends a request to the coordinator; the coordinator determines whether the program is a distributed application through collecting process information of each node and returns the results to them. If the program is not a distributed application, we only need to identify and eliminate duplicate memory contents at the heap and stack page. Otherwise, we need to rely on the information provided by the coordinator to eliminate the redundancy between different process of nodes.

Now we describe the heap and stack data deduplication algorithm in distributed application. First, we need to get the hash value of each memory page by hashing the content of each page. If the hash value exists in the local hash table, we can regard the page as a duplicate and obtain the page index from the hash table and copy it to the checkpoint file. Otherwise, we need to send the information to the coordinator to query whether the hash value exists in other processes.

If the coordinator returns true, we will get the page index from the coordinator and copy it to the checkpoint file. Otherwise, we need to store the page content to the checkpoint file and generate a page index and send it to the coordinator. The index contains the checkpoint file name, the offset of the page in checkpoint file, and the page length. The steps of this algorithm are depicted graphically in Algorithm 1.

```
(1)  function DoDedup(Memorydata, type)
(2)    LocalHashes ← nil
(3)    Mpiflag ← GetProgramType()
(4)    UniqueChunkNum ← 0
(5)    ChunkNum ← 0
(6)    if Mpiflag = false then
(7)      return SingleDodedup(Memorydata, type)
(8)    end if
(9)    while p_i ∈ Memorydata do
(10)     h_i ← Hash(p_i)
(11)     if h_i ∉ LocalHashes then
(12)       UniqueChunkNum + +
(13)       if InquireFromCoor(h_i, type) = true
         then
(14)         DownloadIndexInfo(indexinfo)
(15)         StoreUniqueChunk(indexinfo)
(16)        else
(17)         StoreUniqueChunk(p_i)
(18)         GenerateIndex(indexinfo)
(19)         UploadIndexInfo(h_i, type, indexinfo)
(20)         end if
(21)       MetaData[ChunkNum]          ⟵
     UniqueChunkNum
(22)       ChunkNum ← ChunkNum + 1
(23)       Hashes ← LocalHashes ∪ {h_i}
(24)     else
(25)       MetaData[ChunkNum]          ⟵
     UniqueChunkNum
(26)       ChunkNum ← ChunkNum + 1
(27)     end if
(28)   end while
(29) end function
```

Algorithm 1: Heap and stack data depulication algorthim.

4.3. Dynamic Link Library, Shared Memory, and Code. In a distributed application, all of the dynamic link library, code, and mmap of multiple processes on the same node are the same. For all the running processes, this content only retains one copy. In DMTCP, however, these contents are copied to the local checkpoint file. Therefore, when setting a checkpoint, we just need to copy the pages of the dynamic link library, shared memory, and code to a checkpoint file, and the other checkpoint files only need to save the corresponding index information. However, for single-node applications, the effects of data deduplication are not prominent. So this algorithm does not apply to single-node applications. DMTCP invokes gzip compression by default.

Like Algorithm 1, we first need to query the program type. For a distributed program, we continue to the next step. Otherwise, the algorithm is over. Next, we just have to send local IP address to the coordinator to query whether there are multiple processes running on the local node. If the coordinator receives multiple IP addresses, it means the result is true. The coordinator will send the result to all processes. For shared memory segments, our algorithm is only suitable for multiple processes on a node. In order to eliminate redundancy between all processes dynamic link library and code segment, we send a query to the coordinator to know whether there is an index for the segment on the coordinator. If the index does not exist, we copy the dynamic link library, code, and DLL content to the checkpoint files and then build an index and upload it to the coordinator. Otherwise, the index information is obtained from the coordinator and saved to the checkpoint file. The steps of this algorithm are depicted in Algorithm 2.

4.4. Restart. Let us look at the operation of the stack segment during the restart. For single-node applications, according to the normal process of restart, the difference is that if the data read from the checkpoint file is index, we need to locate and read the real content from the memory according to the index. For distributed applications, the restart is more complicated. Each node containing unique block information needs to create a listener thread, which is used to monitor requests from other processes. After receipt, the requested content will be sent to the requesting process. Before creating the listener, we need to send initialization information to the coordinator for registration. Registration information includes the checkpoint file name, IP address, and port number. When reading the stack and heap information from the checkpoint file, we first read the metadata of each page and locate each page from the meta information. If the page

```
(1)  function DoDedup1(LibInf, type)
(2)    Mpiflag ← GetProgramType()
(3)    pos ← 0
(4)    Ckptname ← Getckptname()
(5)    if Mpiflag = true then
(6)      return SingleDodedup(LibInf, type)
(7)    end if
(8)    if InquireFromCoor(LibInf.name, type) = true then
(9)      Index ← nil
(10)      LibInf.flag ← 2
(11)      GetInfFromCoord(Index)
(12)      LibInf.pos ← Index.pos
(13)      LibInf.pos ← Index.ckptname
(14)      StoreToFile(LibInf)
(15)   else
(16)      LibInf.flag ← 1
(17)      StoreToFile(LibInf, pos)
(18)      StoreToFile(LibInf.data)
(19)      UpLoadToCoord(LibInf, Ckptname, pos)
(20)    end if
(21)  end function
```

Algorithm 2: Dynmic link library, code, and mamap data deduplication algorthim.

exists locally, then we can read the corresponding content to restart. Otherwise, we need to get the content from other nodes. Before sending the request message to other nodes, we need to send a request to the coordinator to get the connection information to other nodes. The steps of this algorithm are depicted in Algorithm 3.

At the checkpoint recovery phase, we need to remap program states into memory. We take the dynamic link library as an example to introduce the recovery phase. When reading the dynamic link library from the checkpoint file, we first obtain the header information of the dynamic link library. We get the checkpoint file name of the dynamic link library and check its existence on the local node. If true, we continue to read the content from the checkpoint file. Otherwise, we need to read the corresponding content from other nodes. Like the previous recovery operation, we need to request the coordinator to obtain information about the checkpoint file and finally retrieve its content to restart. When the content of the checkpoint file is read and remapped into memory, the program can be restored to a correct state to run.

5. Experiment

In this section, we evaluate our approach on QingCloud, which is a public cloud service, providing on-demand, flexible, and scalable computing power. In particular, QingCloud is a popular cloud platform, and, due to its per-second charge method, our experiment will be conducted on QingCloud. For simple comparisons, we use two nodes on QingCloud to conduct our experiments. The configuration of the cloud node is dual-core processors and 2 GB of RAM. The system ran 64-bit ubuntu 14.04. The two nodes are connected by LAN. Experimental programs are divided into two categories: single-node programs and distributed programs across the nodes of a cluster.

For single-node programs experiments, we use the following applications: Python (2.7.6) an interpreted, interactive, object-oriented programming language; Vim (7.4) interactively examining a C program; BaseLine, a simple C application, whose function is to print a 32-bit integer value every one second; BC (1.06.95) an arbitrary precision calculator language; Emacs (2.25) a well known text editor; and Perl (5.18.2) Practical Extraction and Report Language interpreter. To show the breadth, we present checkpoint times, restart times, and checkpoint sizes on a wide variety of commonly used applications. These results are presented in Figures 2, 3, and 4.

For distributed programs, the checkpoint will be written to local disk. We use the MPI package for our experiments. We use the following programs to implement our experiments: NAS NPB3.3 and BaseLine. The BaseLine is a simple MPI program that calculates Pi. The NAS Parallel Benchmarks (NPB) are a small set of programs designed to help evaluate the performance of parallel supercomputers. Problem sizes in NPB are predefined and indicated as different classes. The benchmarks run under MPICH2 are BT (Block Tridiagonal, level C), SP (Scalar Pentadiagonal, level C), EP (Embarrassingly Parallel, level C), LU (Lower-Upper Symmetric Gauss-Seidel, level C), MG (Multigrid, level C), and IS (Integer Sort, Level C).

```
(1)  function UndedupHeapAndStack(fd)
(2)    Mpiflag ← GetProgramType()
(3)    AreaInf ← nil
(4)    NodesInf ← nil
(5)    NodesFd ← nil
(6)    if Mpiflag = true then
(7)      CreateServerThread()
(8)      Register()
(9)      GetNodesInfFromCoor(NodesInf)
(10)   else
(11)     return SingleUnDodedup(fd)
(12)   end if
(13)   while ReadArea(fd, AreaInf) do
(14)     if AreaInf.ckptname = MyCkptName() then
(15)       ReadAndRemap(fd, AreaInf)
(16)     else
(17)       if NodeFd[AreaInf.ckptname] ≠ nil then
(18)         ConnFd ← 0
(19)         ConnFd ←
       ConnectNode(NodesInf, AreaInf.ckptname)
(20)         NodesFd[AreaInf.ckptname] ←
       ConnFd
(21)       RemapFromOtherNode(NodesFd, AreaInf)
(22)     end if
(23)   end if
(24)  end while
(25)  end function
```

Algorithm 3: Restart.

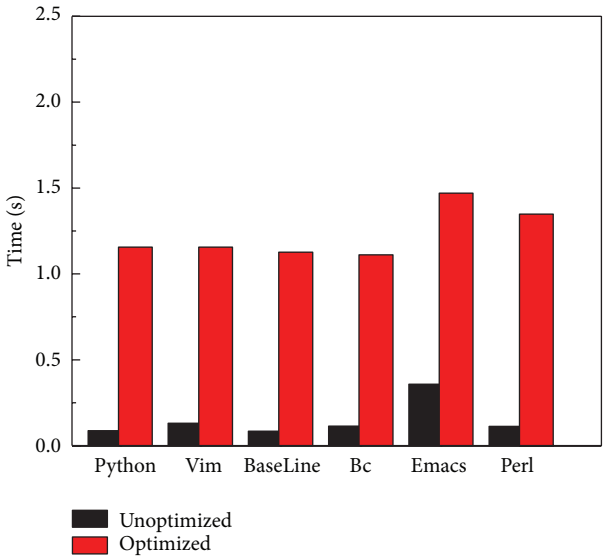


Figure 2: Checkpoint time of single-node programs.

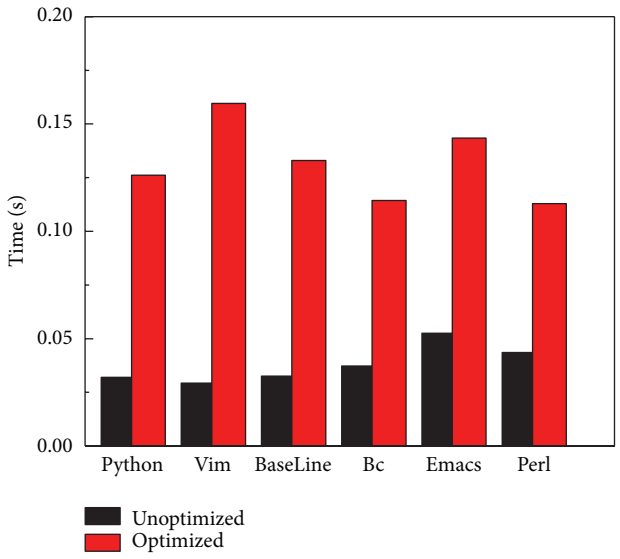


Figure 3: Single-node programs restart time.

We report checkpoint times, restart times, and checkpoint file sizes for a suite of distributed applications. In each case, we report the time and file size when no compression is involved. The experiment was repeated five times. In Figure 7, the checkpoint file size includes the sum of the checkpoints on the two nodes.

6. Experimental Analysis

The graphs in Figure 2 show that, as compared to the original DMTCP, the optimized version increases the checkpointing time. Using the data deduplication algorithm and checking the application category are two main reasons for the increase of time consumption. The checkpointing times under the

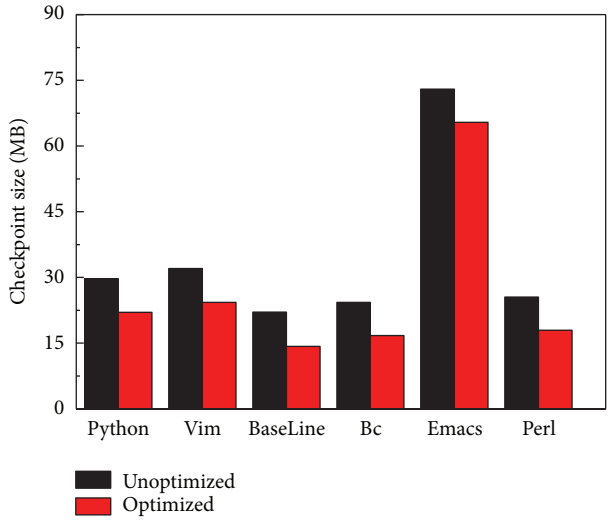


Figure 4: Single-node programs checkpoint file size.

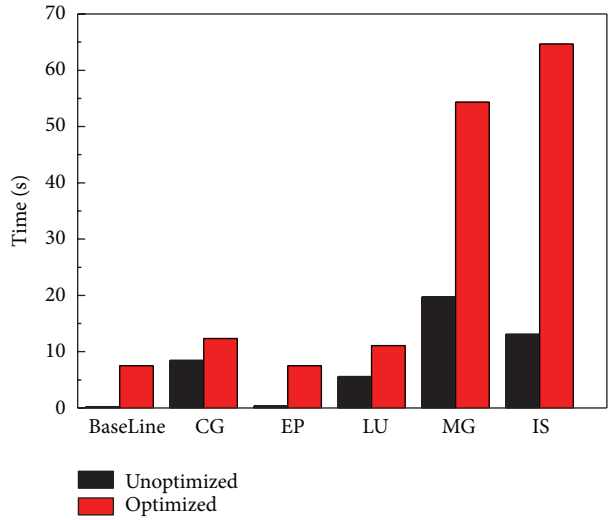


Figure 5: Distributed programs checkpoint time.

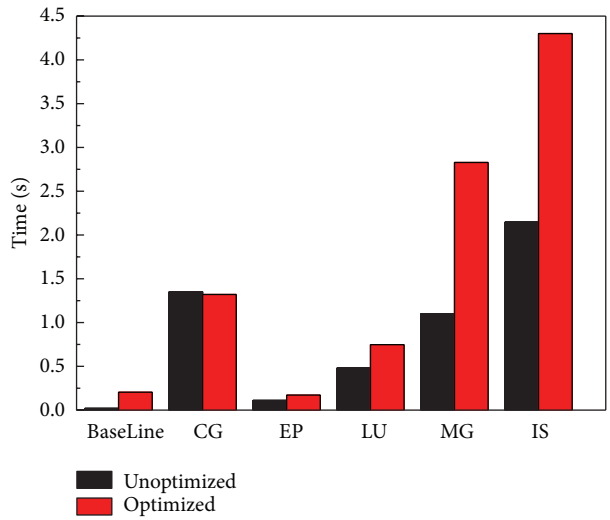


Figure 6: Distributed programs checkpoint restart time.

original DMTCP are 0.08, 0.13, 0.08, 0.11, 0.35, and 0.11 seconds for Python, Vim, BaseLine, Bc, Emacs, and Perl, respectively. As a comparison, for the optimized DMTCP, the checkpointing time increases, such as 1.15 seconds for Python, 1.56 seconds for vim, 1.12 seconds for BaseLine, 1.11 seconds for Bc, 2.9 seconds for Emacs, and 1.34 seconds for Perl.

Figure 3 presents the restart time. In the restart time, For the original DMTCP, the time consumption for the restart operation is 0.03, 0.02, 0.03, 0.03, 0.05, and 0.04 seconds for Python, Vim, BaseLine, Bc, Emacs, and Perl, respectively. As in the case of the checkpoint phase, the restart time also increases. The restart time for each application is increased by about 0.1 seconds, in order to check the type of application to communicate with the coordinator, which resulted in additional costs.

In Figure 4, we can see that the checkpoint file size of all applications is different between the original DMTCP and optimized DMTCP. For single-node programs, we only remove redundant data in the stack, but the final effect is still good. Python's checkpoint file size using the original DMTCP is 29 M and the file size for the optimized version is only 21, indicating about 25% saving. Similarly, the file sizes of other programs in optimized DMTCP are all reduced nontrivially. For example, Vim is reduced by 24%; BaseLine is reduced by 35%; Bc is reduced by 31%; Emacs is reduced by 10%; Perl is reduced by 29%.

Figures 5, 6, and 7 show the results for distributed applications. Similar to the single-node programs, the checkpoint time when using the optimized DMTCP increases for all of the applications. For example, for the case of CG, the checkpoint time is 8 seconds versus 12 seconds. For other test programs, the time is also increased. For example, BaseLine increases 7.2 seconds, EP increased 7 seconds, and LU increased 5.5 seconds. The cost is mainly incurred by the operation of data deduplication. Restart time also increases, and the quantity depends on the duplicate data volume of the various applications.

Figure 7 shows all the checkpoint file sizes of the distributed program. In the original DMTCP, the checkpoint sizes for the testing programs range from 65 MB to 1607 MB. Correspondingly, in the optimized DMTCP, the checkpoint size for BaseLine, CG, EP, LU, MG, and IS is 34 M, 564 M, 36 M, 396 M, 47 M, and 800 M, respectively, indicating a checkpoint size reduction by 47.7%, 49.8%, 47.7%, 49%, 50.4%, and 50%.

Through the above experiments, we can conclude that, by using our method, despite the increase of checkpoint and restart time, the checkpoint file size can be greatly reduced. For the distributed applications, the effect is much more prominent. In the experiment, single-node program mainly reduced the stack segment of redundant information. But, for distributed applications, we reduce the redundant information in the heap and stack segments between different processes. Moreover, the code, dynamic link libraries, and the

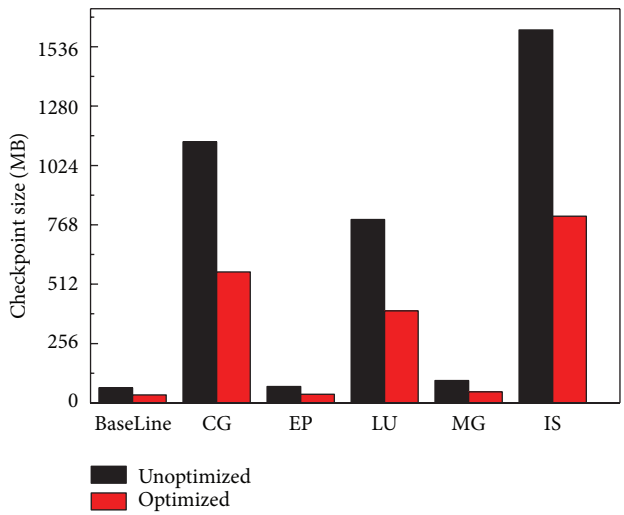


Figure 7: Distributed programs checkpoint file size.

contents of shared memory in the same node are stored in the checkpoint file of a process, while the remaining checkpoint files are only stored in the content index. This is the effect of the experimental program distributing better the single-node program.

In this paper, the experimental program will be divided into two types: single-node program and distributed program. When checking the type of the program, process of each node needs to wait until the coordinator collects information of all processes, which will lead to some of the processes being blocked. When the checkpoint is set, the heap and stack segment uses data deduplication technology, which will be segmented data block. In the distributed application, the data deduplication technology will communicate with the coordinator to query block information. In the checkpoint restart phases, each node containing unique block information needs to create a listener thread, which is used to monitor requests from other processes. Above all, these operations will make extra time overhead; of course, there are some other operations that will also have time overhead. We plan to improve our approach to reduce the time overhead in the future.

7. Conclusions and Future Work

In this paper, we conduct a detailed analysis about the data redundancy in checkpoint files and the potential of utilizing this finding to optimize checkpointing systems. Based on the findings, we propose the design and implementation of a system, which leverages inline data deduplication to achieve the goal of reducing the size of checkpoint file. We perform extensive experiments on a wide range of single-node and distributed applications, and the results demonstrate the effectiveness of our system that is more prominent for distributed applications. However, the results also indicate that there are rooms for improvement in time consumption, which we plan to address in future work.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This research was supported in part by the National Science Foundation of China under Grants 61272190 and 61572179, the Program for New Century Excellent Talents in University, and the Fundamental Research Funds for the Central Universities of China.

References

[1] M. Armbrust, A. Fox, R. Griffith et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: high availability via asynchronous virtual machine replication," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pp. 161–174, San Francisco, Calif, USA, 2008.

[3] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[4] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375–408, 2002.

[5] S. Agarwal, R. Garg, M. S. Gupta, and J. E. Moreira, "Adaptive incremental checkpointing for massively parallel systems," in *Proceedings of the 18th Annual International Conference on Supercomputing*, pp. 277–286, ACM, July 2004.

[6] N. Naksinehaboon, Y. Liu, C. Leangsuksun, R. Nassar, M. Paun, and S. L. Scott, "Reliability-aware approach: an incremental checkpoint/restart model in HPC environments," in *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)*, pp. 783–788, IEEE, Lyon, France, May 2008.

[7] K. B. Ferreira, R. Riesen, P. Bridges, D. Arnold, and R. Brightwell, "Accelerating incremental checkpointing for extreme-scale computing," *Future Generation Computer Systems*, vol. 30, no. 1, pp. 66–77, 2014.

[8] J. S. Plank, Y. Chen, K. Li, M. Beck, and G. Kingsley, "Memory exclusion: optimizing the performance of checkpointing systems," *Software—Practice and Experience*, vol. 29, no. 2, pp. 125–142, 1999.

[9] D. Ibtesham, D. Arnold, K. B. Ferreira, and P. G. Bridges, "On the viability of checkpoint compression for extreme scale fault tolerance," in *Euro-Par 2011: Parallel Processing Workshops*, pp. 302–311, Springer, 2012.

[10] X. Lin, G. Lu, F. Douglis, P. Shilane, and G. Wallace, "Migratory compression: coarse-grained data reordering to improve compressibility," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST '14)*, pp. 257–271, USENIX Association, 2014.

[11] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST '13)*, pp. 183–198, San Jose, Calif, USA, February 2013.

[12] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08)*, article 18, USENIX Association, 2008.

[13] L. Valiant, *Center for Research in Computing Technology*, Harvard University, Cambridge, Mass, USA, 1994.

[14] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani, "Demystifying data deduplication," in *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion*, pp. 12–17, ACM, Leuven, Belgium, 2008.

[15] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, "iDedup: latency-aware, inline data deduplication for primary storage," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST '12)*, vol. 12, pp. 1–14, San Jose, Calif, USA, February 2012.

[16] B. Agarwal, A. Akella, A. Anand et al., "Endre: an end-system redundancy elimination service for enterprises," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI '10)*, pp. 419–432, 2010.

[17] M. Litzkow and M. Solomon, *Supporting Checkpointing and Process Migration Outside the Unix Kernel*, 1992.

[18] J. Duell, *The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart*, Lawrence Berkeley National Laboratory, Berkeley, Calif, USA, 2005.

[19] J. Ansel, K. Arya, and G. Cooperman, "DMTCP: transparent checkpointing for cluster computations and the desktop," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS '09)*, pp. 1–12, IEEE, Rome, Italy, May 2009.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 4516596, 12 pages
http://dx.doi.org/10.1155/2016/4516596

Research Article

Research on Linux Trusted Boot Method Based on Reverse Integrity Verification

Chenlin Huang,[1] Chuanwang Hou,[1] Huadong Dai,[1] Yan Ding,[1] Songling Fu,[2] and Mengluo Ji[3]

[1]*School of Computer, National University of Defense Technology, Changsha, Hunan 410073, China*
[2]*College of Polytechnic, Hunan Normal University, Changsha, Hunan 410073, China*
[3]*Department of Computer and Information Engineering, Luoyang Institute of Science and Technology, Luoyang, Henan 471023, China*

Correspondence should be addressed to Huadong Dai; daihuadong@kylinos.cn

Received 25 February 2016; Accepted 8 May 2016

Academic Editor: Florin Pop

Trusted computing aims to build a trusted computing environment for information systems with the help of secure hardware TPM, which has been proved to be an effective way against network security threats. However, the TPM chips are not yet widely deployed in most computing devices so far, thus limiting the applied scope of trusted computing technology. To solve the problem of lacking trusted hardware in existing computing platform, an alternative security hardware USBKey is introduced in this paper to simulate the basic functions of TPM and a new reverse USBKey-based integrity verification model is proposed to implement the reverse integrity verification of the operating system boot process, which can achieve the effect of trusted boot of the operating system in end systems without TPMs. A Linux operating system booting method based on reverse integrity verification is designed and implemented in this paper, with which the integrity of data and executable files in the operating system are verified and protected during the trusted boot process phase by phase. It implements the trusted boot of operation system without TPM and supports remote attestation of the platform. Enhanced by our method, the flexibility of the trusted computing technology is greatly improved and it is possible for trusted computing to be applied in large-scale computing environment.

1. Introduction

With the boom of Internet, the lack of a trustworthy infrastructure has been a barrel for the healthy development of modern computing systems. More and more threats are introduced due to the design flaws in software and hardware, the improper authorization and authentication for legal users, the abusing use of resources, and so forth. The key to solve these problems is to build a trustworthy computing environment, where the safety of end system is well designed and can be verified and trusted. Trusted computing technology proposed by the Trusted Computing Group (TCG) is one of the main practical efforts to achieve this goal. Trusted computing architecture is based on the trusted hardware, Trust Platform Module (TPM), and realizes transitive trust through the constant trust metric in the progress of system boot process to build a trusted computing environment.

Trusted platform module TPM and the related software are introduced in trusted computing platform technology to be as trusted root of the system, through the trust transfer process to ensure the credibility of computing platforms and applications and to improve the security of the terminal platform. However, in order to support a variety of security features in TCG specifications, a special trusted hardware TPM is required to be deployed in the mainboard, which has become a main barrier limiting the popularization of the trusted computing platform technology. TPM is the base of the trust chain and the trusted root throughout the trusted boot process, which records and transfers trusted states in end system. However, the TPM chips are not yet widely deployed in most computing devices so far, thus limiting the applied scope of trusted computing technology. It is almost impossible to implement an overall trusted network computing environment due to the hardware barrel.

To fix the problem of lacking trusted hardware in existing computing platform, an alternative security hardware, USBKey, is introduced in this paper to simulate the basic functions of TPM and a new reverse USBKey-based integrity verification model is proposed to implement the reverse integrity verification of the operating system boot process, which can achieve the effect of trusted boot of the operating system in end systems without TPMs.

We have designed and implemented a Linux trusted boot method based on reverse integrity verification, with which the integrity of data and executable files in the operating system are verified and protected during the trusted boot process phase by phase. It implements the trusted boot of operation system without TPM and supports remote attestation of the platform. Enhanced by our method, the flexibility of the trusted computing technology is greatly improved and makes it possible to be applied in large-scale computing environment.

2. The Trusted Boot in Operating Systems

2.1. Related Works. The trusted hardware, TPM (Trusted Platform Module), plays a key role in trusted computing, working as the base of trusted computing architecture and the core to enhance the credibility of the general-purpose computing platforms and networks. At present, the core standard is TPM 2.0 [1]. TPM functions as a trusted root of trusted computing platform, providing key cryptographic functions and protected storage space which are necessary to build trusted computing environment by coordinating with other trusted computing software and hardware.

Given the limitations in TPM's architecture and cryptographic algorithms, new trusted computing architectures Trusted Cryptography Module (TCM) [2] and Trusted Platform Control Module (TPCM) [3] have been proposed by scholars in China as official standards. Double certificate structure is designed in TCM, and Chinese government approved cryptographic algorithms are supported besides commercial ones. TPCM is capable of performing active control and trusted measurement on hardware level, which controls the trusted computing base (TCB) in end systems. Compared with TPM, a more rigid and trustworthy architecture is implemented in TPCM.

Recently, trusted mobile computing platforms and trusted cloud services are research focuses in both industry and academia. Studies relating to trusted mobile computing platform focused on TrustZone in smart devices, trusted identity management, privacy protection, and other aspects. Ekberg et al. propose ObC (On-board Credentials) system which allows third party to develop and deploy credentials in the device based on TrustZone and provides a TEE (Trusted Execution Environment) function for application developers [4]; Sujeen and Periasami propose a data security and privacy protection technology based on TPMs [5]. Nyman et al. propose a new, rich authorization model to solve the traditional eID management problem by enhancing platform integrity verification and eID authentication based on TPM 2.0 [6]. Zhao et al. implement and evaluate trusted boot of the Trusted Execution Environment (TEE) based on ARM TrustZone with the on-chip SRAM Physical Unclonable Functions (PUFs) [7]. Santos et al. extend the concept of trusted computing to the background of Infrastructure as a Service (IaaS) and propose a trusted cloud computing platform (TCCP) for ensuring the confidentiality and integrity of computations [8]. But, users of cloud computing do not have currently appropriate tools for their verification of confidentiality, privacy policy, computing accuracy, and data integrity. Banirostam et al. propose Trusted Cloud Computing Infrastructure (TCCI) providing a closed execution environment for infrastructure service developers by a User Trusted Entity (UTE) [9]. Habib et al. propose a multifaceted trust management system architecture for cloud computing marketplaces and related approaches [10].

Since not all platforms are equipped with TPM module, the traditional TPM leads to the loss of mobility and flexibility due to the binding of user's identity with specific trusted platform. Accordingly, some scholars introduce some USBKey-based secure boot solutions based on the principles of separating platform and user certificate. To prevent the master boot record from easily being manipulated and infiltrated by bootkits, Müller et al. present Stark, which mutually authenticates the computer and the user in order to resist keylogging during boot and implements trust bootstrapping from a secure token (a USB flash drive) [11]. Based on UEFI (Unified Extensible Firmware Interface), Kushwaha proposes the creation of ESP (EFI System Partition) on the USBKey [12]. Since the ESP contains the boot loader and other critical codes for booting, the system always needs the USBKey to boot the system to a running state. Meanwhile, Microsoft's secure boot architecture Win 8 increases UEFI-based secure boot components and further enhances security on the traditional concept of trusted booting.

In summary, the more flexible and practical trusted boot technologies in operating system are being developed to be bound with new user application schemas and scenarios and to suit the development of mobile computing and cloud computing.

2.2. Trusted Boot in Linux. Trusted boot is one of the core functions of trusted computing platform. With the support of trusted hardware, a trusted running environment for services and applications is built with the verification of the integrity of the whole hardware and software during system boot process.

The following three key points must be guaranteed during the trusted boot process.

(1) The chain of trust must be established sequentially. Before the transference of control rights, the executable entity must be measured by trusted computing base. It can only be loaded and gain control rights after its integrity is verified, which fulfills the procedure of establishing chain of trust. (2) All the metrics and calls involved in the process of the establishment of the trust chain will eventually be completed by TPM. (3) During the establishment of chain of trust, all important secret data involved including keys, premeasurement data, and verification data must be stored and sealed inside TPM. TPM is responsible for ensuring the integrity and confidentiality of the secret data. Unlike

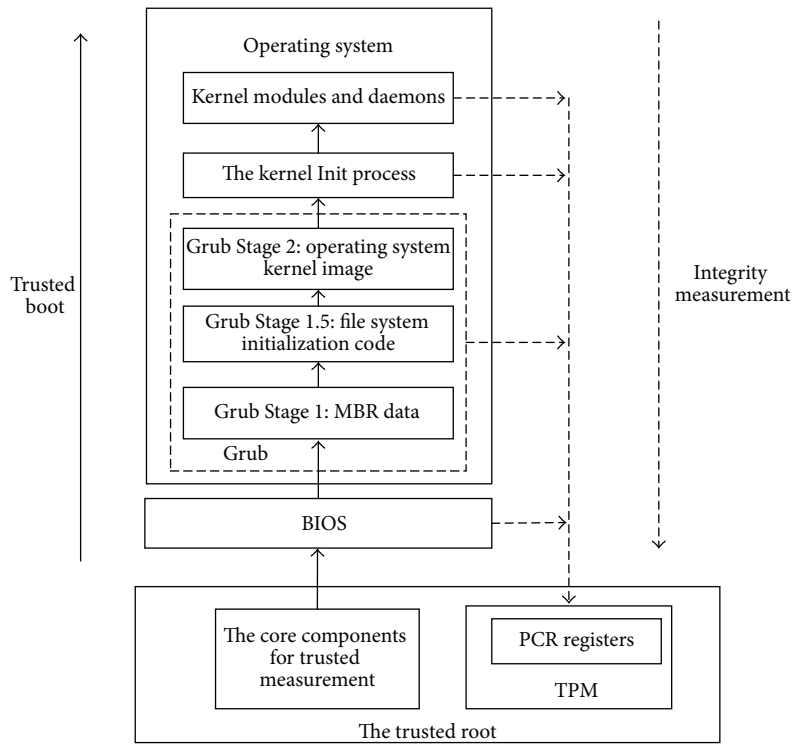


Figure 1: Linux trusted boot model.

removable storage devices or memory, there is no external call interface provided to access the secrete data in TPM, which ensures its confidentiality and credibility.

Taken the Linux trusted boot process based on TPM as an example, the trusted boot model in operating system is shown in Figure 1.

Trusted boot mainly includes two phases: the boot of hardware platform and the startup of operating system. The boot of hardware platform starts from the platform power on to the BIOS initialization and ends after the BIOS passes control rights to the boot loader. The reliability of hardware environment is measured and verified in this phase. The startup phase of the operating system begins with the loading of operating system loader from the main boot sector, and then the operating system kernel is loaded and ends till the running of the Init process. This stage is mainly responsible for checking the creditability of the system startup process and the operating system kernel. The trusted boot process of the startup phase of the operating system based on TPM is as follows.

Step 1. Trusted BIOS loads boot loader stored in Boot sector and then sends it to TPM to be measured and verified. Once TPM has verified its integrity, the boot program is loaded to memory 0000:7C00h, and then the BIOS passes control rights to the CPU to run the Boot program to further load operating system.

Step 2. TPM validates the operating system loader program, such as Grub in Linux. If the verification is successful, the Grub Stage 1 code in the master boot sector is loaded into memory and gains the trusted boot control to further load operating system kernel.

Step 3. The Grub Stage 1 continues trusted boot process by first validating Grub Stage 1.5 code with TPM, if it is successful, loads, and runs the code of the Stage 1.5 phase. At the end of this stage, the file system is mounted.

Step 4. The Grub Stage 2 code is verified by TPM and loaded by trusted Grub Stage 1.5. After successfully gaining control, it will verify the integrity of the configuration file "/boot/Grub/Grub.conf" in which the locations of the disk partitions, the kernel image, and virtual RAM disk file initrd are recorded.

Step 5. The Grub Stage 2 code opens the configuration file, reads the operating system kernel image, and tries to verify the integrity of the operating system kernel image by TPM. If it is successful, the operating system kernel image is loaded and gains control.

Step 6. Once the operating system kernel image is loaded, TPM will measure and verify the Init process. If the Init process is trusted, the kernel key data structures will be created and the kernel Init process will be loaded and take control.

Step 7. Firstly, the Init process determines the list of the kernel modules needed to be loaded and the daemons needed to

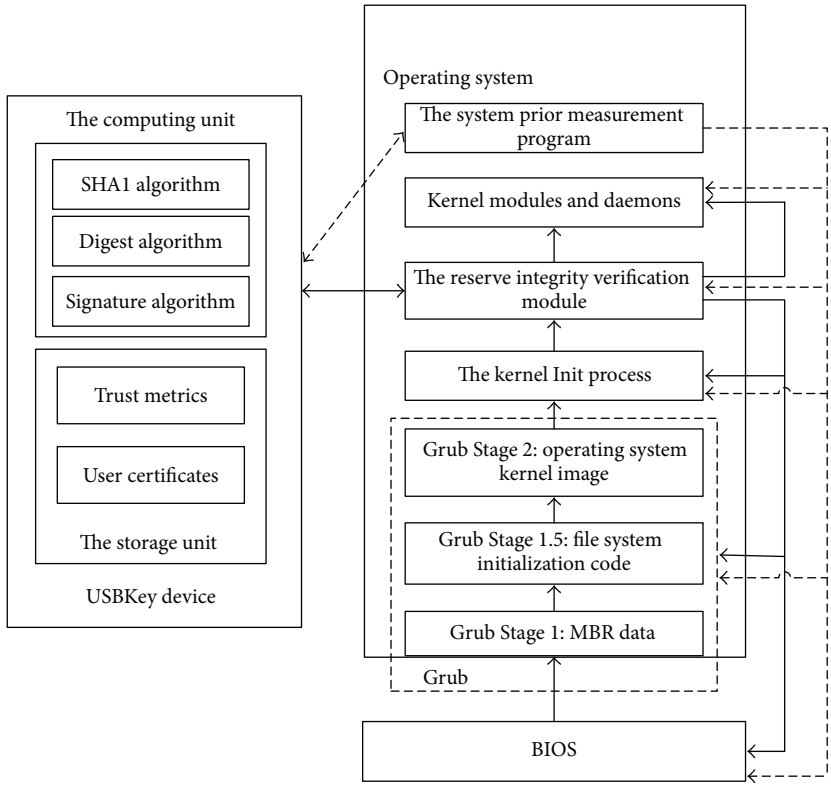


Figure 2: The reverse integrity verification model based on USBKey.

be created based on the system configuration. Then, it will measure and verify each kernel module and daemon with TPM module before they are loaded. Only the trusted kernel modules and daemons are run sequentially to guarantee that the initialized computing environment is trusted. At last, the Init process starts receiving users' inputs, and a trusted computer is ready to be used.

At this point, the trusted operating system boot process is over. As we can see, the traditional trusted boot process based on TPM is in forward direction, which means all measurements and verifications are strictly consistent with the operating system boot process. The chain of trust is established in a strict sequence.

3. A Linux Trusted Booting Method Based on Reverse Integrity Verification

3.1. A Reverse Integrity Verification Model with USBKey. As shown in Figure 2, in this paper, the operating system trusted boot method based on reverse integrity verification is implemented by the coordination of operating system kernel, the BIOS (firmware), and USBKey (USB smart card). The operating system kernel provides a basic software system running environment, including the drivers for system hardware and the construction of the system execution environment. In the process of the system boot, there are four parts in operating system kernel which are system boot Stage 1, Stage 1.5, Stage 2, and kernel modules. The BIOS firmware covers the initial stage of system boot, completes the initialization of hardware, and passes control to the boot loader. USBKey used in this method has built-in CPU, memory, Chip Operating System (COS), and internal safe data storage units, where secret data are stored, such as the user digital certificates and secret keys. There is also a sealed computing unit inside USBKey that supports cryptographic operations such as SHA1 algorithm, signature, authentication, data encryption and decryption, and data digest. All operations are calculated in the COS of USBKey which is totally safe to the outside world. As a trusted hardware, USBKey provides us a method to validate the integrity of data reliably.

Working as a trusted root, TPM provides support for the storage of trusted measurement and trusted state (PCR) in the establishment of chain of trust during trusted boot. It is also the starting point and foundation of system trusted boot and trust measurement. Especially in TPCM framework, the trusted hardware is regarded as trusted base and is the first functional hardware after powering on the system, so as to provide guarantee to keep the whole system in trusted states.

Similar to the TPM, USBKey, as widely used security hardware, also has built-in security guarantee, built-in trusted measurement algorithms, and built-in secure storage space. But in order to implement trusted measurement as TPM module does, USBKey still needs to get rid of the following two limitations: (1) Hardware driver needs to be loaded before USBKey could be used, which makes it

unable to activate trusted measurement from powering on the system as required in TPCM. (2) There are no PCRs in the USBKey built-in storage. New software data structures have to be designed to simulate PCRs in TPM, so as to record system's trusted states and support trusted measurement during trusted boot.

To handle the first problem, we assume that a trusted prior measurement could be preceded by the kernel trusted measurement module while the system was in the initial trusted state. The trusted prior measurement generates foundation trust metrics for trusted measurement. To solve the second problem, a set of data structures are defined to simulate PCR registers in TPM which can be used to store the measurement values in the process of prior measurements and trusted boot. All measurement data are stored in safe storage area in USBKey.

The reverse integrity verification model based on USBKey is shown in Figure 2, in which the dotted line shows the process of the prior measurement and the solid line shows the process of the reverse integrity measurement in trusted boot process. There are five key elements in a trusted measurement model based on TPM: *PCR values for prior measurement, TPM, targets to be measured, PCR values, and verification results*. First of all, a *trust metric base* needs to be constructed as a credential reference library for later trusted boot in the prior measurement phase. During prior measurement, the entities in the system are measured by TPM following the sequence of system boot and the trust measurement values in PCRs are recorded into trust metric base. Then, in the process of system boot, TPM will measure the entities to be loaded, which are targets to be measured sequentially and compare the PCR values with the PCR values for prior measurement for consistency to reach the verification results.

In the reverse integrity verification model based on USBKey, the five key elements are *trust metrics for prior measurement, USBKey, targets to be measured, trust metrics, and verification results*. In our model, trust metrics refer to the trust measurement values generated by USBKey. Compared with the traditional trusted measurement model based on TPM, the key elements are similar, while the TCB and the trust measurement procedure differ a lot. The combination of "USBKey + trust metric base" functions as the root of trust instead of TPM only. The process of trusted boot is also divided into two phases: *the reverse integrity verification phase* and *the trusted boot phase*. After system starts up and successfully drives and loads USBKey, *the Reserve Integrity Verification Module* will initialize the reverse integrity verification phase. It will access the loaded system entities and measure and verify their integrity values with USBKey by comparing with the *trust metrics for prior measurement*. After phase one, a reliable system environment is verified and the trust foundation of the current system state is established. Then, *the trusted boot phase* will handle the rest of system boot the same as the process of trusted boot. *Together with USBKey*, *the Reserve Integrity Verification Module* functions as a TCB and makes sure the following loaded entities are trusted. Once an entity needs to be loaded, the module will call USBKey to measure the entity and verify its integrity by comparing with the *Trust Metrics for Prior Measurement*. Since only trusted entities are to be loaded and run, the whole system will be in a trusted state persistently.

Operating system trusted boot method based on reverse integrity verification contains the following steps.

(1) Prior Measurement. The initial system state can be assumed to be trusted. To start a prior measurement process, a regular system boot will be executed until starting to receive the input of user. Then, the System Prior Measurement Program will be loaded by the Init process and take control. The System Prior Measurement Program starts revising the whole boot process by measuring each stage during system boot sequentially. For every stage, the integrity of the relative entities is measured by USBKey; the measurement values are stored into USBKey safe storage unit to form *trust metric base*.

(2) Reverse Integrity Verification. For regular system boots after prior measurement, the reverse integrity verification can be performed by *the Reserve Integrity Verification Module* and USBKey. Once the operating system kernel is loaded and the USBKey is enabled, *the Reserve Integrity Verification Module* will be loaded into kernel and control the rest of trusted boot. The design of *the Reserve Integrity Verification Module* is similar to trusted software stack, and it is able to complete the integrity measurement of the target by cooperating with USBKey. First, it will read the prior measurement values for each boot stage from trust metric base, which are reference library for later trust verification. Then, each stage during system boot will be measured by USBKey. By referencing the trust metric base, the current integrity state of the system is verified by comparing the current measurement value with the relative prior measurement value. If all the loaded boot stages have passed reverse integrity verification, the current system is marked as in a trusted state. *The Reserve Integrity Verification Module* will continually be in charge to guarantee that all the modules, services, and applications loaded later are measured and verified by USBKey. Only trusted entities are to be loaded and run until the trusted boot succeeds and a trusted computing environment is established.

Following the system boot order, there are nine stages of data that are measured and verified in our method:

(1) The BIOS.

(2) The Grub Stage 1 data in the master boot sector.

(3) Grub Stage 1.5 data.

(4) Grub Stage 2 data.

(5) Grub configuration file.

(6) The kernel image file.

(7) The Init process data.

(8) The kernel modules to be loaded by the Init process based on the system configuration.

(9) The daemons to be loaded by the Init process based on the system configuration.

The nine stages can be divided into two parts: BIOS boot and operating system boot. The BIOS image needs to be measured and protected which includes codes for POST

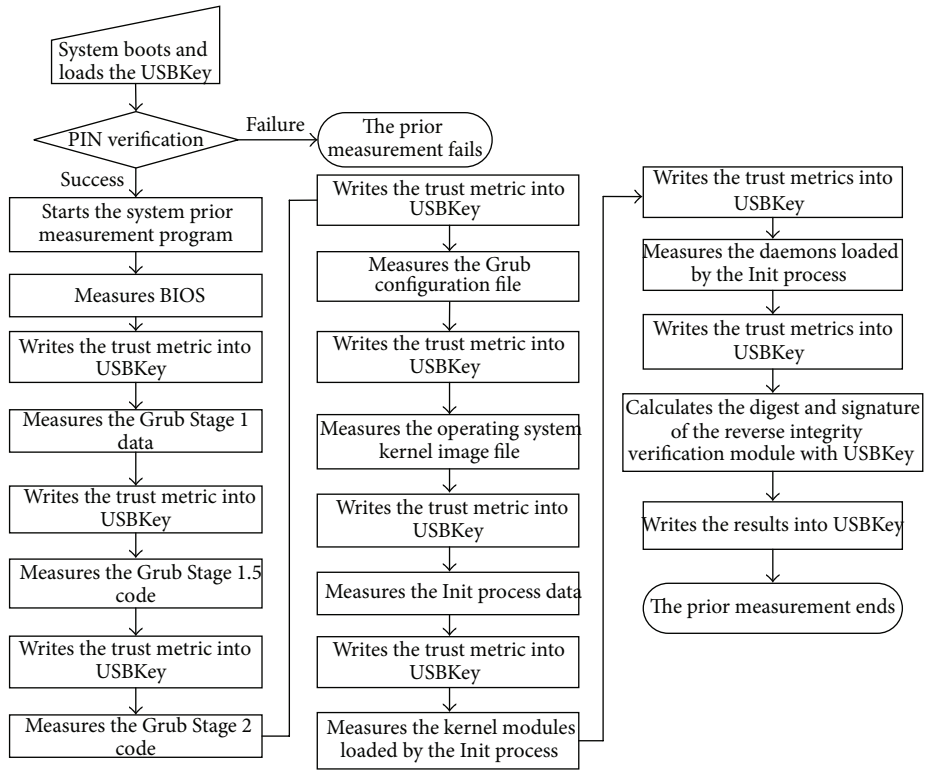


Figure 3: The design of the prior measurement for the reverse integrity verification.

(Power-On Self-Test), system hardware initialization, and loading operating system boot loader. The operating system boot phase contains Grub Stage 1, Grub Stage 1.5, Grub Stage 2, the kernel modules and daemons loaded by the Init process.

In the method, the System Prior Measurement Program and the Reverse Integrity Verification Module are located in the operating system kernel, whose legitimacy and integrity are protected by USBKey signature. They can be loaded if and only if USBKey PIN verification and signature authentication are success. In this paper, SHA1 algorithm in USBKey is applied to perform trust measuring.

3.2. The Design of Prior Measurement for Reverse Integrity Verification. The different stages of system boot are premeasured by USBKey. The trust metric base is constructed by collecting system prior measurement data, which are used to support trust measurement with USBKey in the process of the trusted boot. The design of the prior measurement for the reverse integrity verification is shown in Figure 3, the stages in the process of the operating system boot are measured by USBKey and the prior measurement values are stored in safe storage unit.

In prior measurement phase, a regular system boot will be executed until starting to accept the user input. Then, the System Prior Measurement Program will be loaded and begin measuring each stage of system boot sequentially. The trust measurement values are calculated by USBKey and are stored into USBKey safe storage unit. The trust metric base is constructed after prior measurement phase. As shown in Figure 3, the specific steps are as follows:

(1) Operating system boots up until receiving user input, that is, from powering on the system to the time the Init process is successfully running. Then, it inserts USBKey and verifies user's PIN. If successful, *the System Prior Measurement Program* is verified and loaded; otherwise the prior measurement fails.

(2) *The System Prior Measurement Program* reads the BIOS information and calls USBKey to execute trust measurement with SHA1 algorithm. The returned BIOS trust metric is stored in the safe storage unit in USBKey.

(3) *The System Prior Measurement Program* reads the Grub Stage 1 data from master boot sector and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(4) *The System Prior Measurement Program* reads the Grub Stage 1.5 code and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(5) *The System Prior Measurement Program* reads the Grub Stage 2 code and calls USBKey to execute trust measurement with SHA1 algorithm. The returned

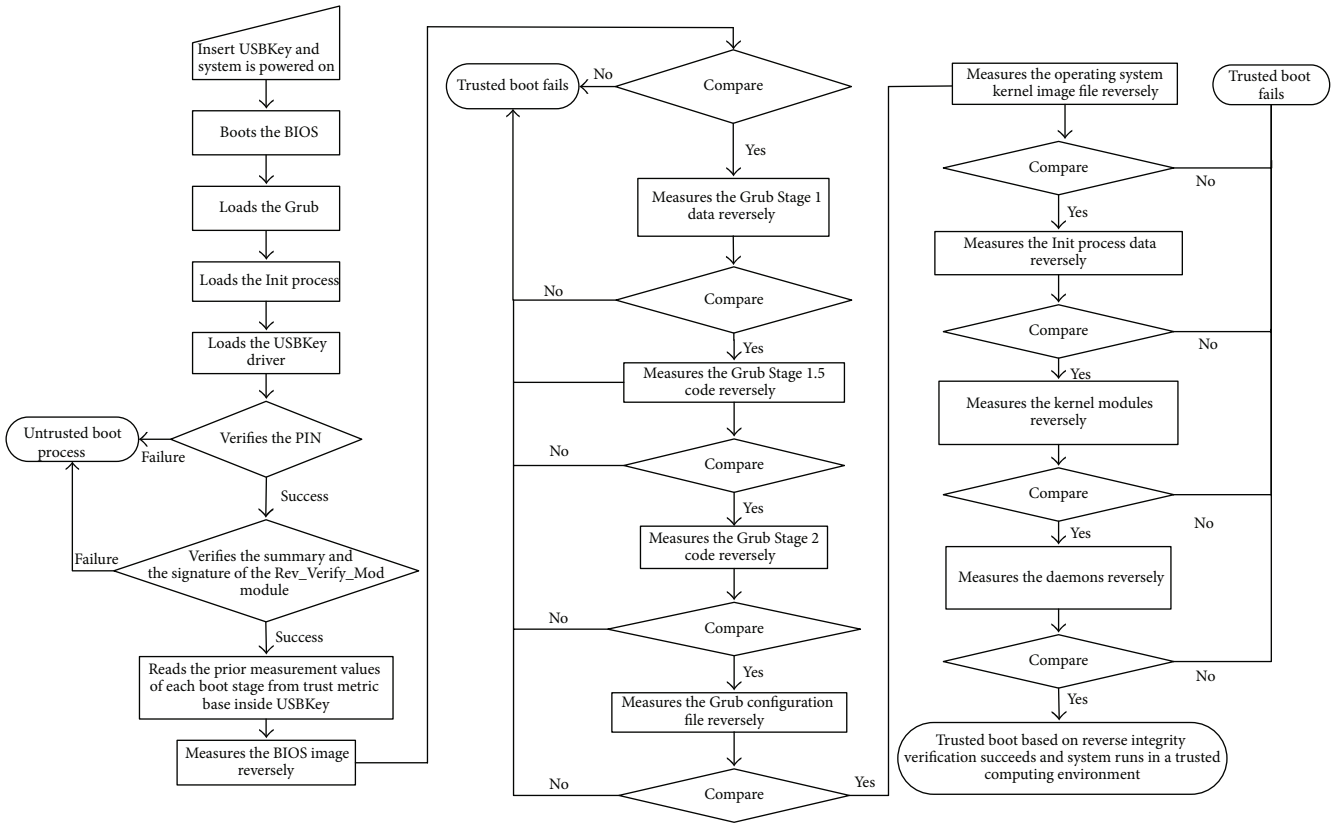


Figure 4: The trusted boot of operating system with reverse integrity verification.

trust metric is stored in the safe storage unit in USBKey.

(6) *The System Prior Measurement Program* reads the Grub configuration file "/boot/Grub/Grub.conf," and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(7) *The System Prior Measurement Program* reads the operating system kernel image file and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(8) *The System Prior Measurement Program* reads the Init process data and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(9) *The System Prior Measurement Program* reads the kernel modules to be loaded by the Init process sequentially and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metrics are stored in the safe storage unit in USBKey.

(10) *The System Prior Measurement Program* reads the daemons to be loaded by the Init process sequentially and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metrics are stored in the safe storage unit in USBKey.

(11) To protect TCB in the method, USBKey precedes the summary and signature verification on the Reverse Integrity Verification Module Rev_Verify_Mod with user digital certificate inside. The results as Rev_Verify_Mod's trust metrics are deposited in the safe storage unit in USBKey.

(12) The prior measurement process comes to an end.

3.3. The Trusted Boot of Operating System with Reverse Integrity Verification. After the completion of prior measurement, USBKey can reverse and verify the operating system booting process according to the stored prior metric values. To support USBKey-based verification, an operating system kernel module, *the Reverse Integrity Verification Module (Rev_Verify_Mod)*, is implemented to measure and verify the entities loaded and to be loaded modules and the modules by calling the USBKey. To demonstrate the detailed USBKey-based reverse integrity verification process, the USBKey-based Linux operating system trusted boot process is shown in Figure 4.

After the prior measurement, the trusted boot is enabled. The system is powered on and operating system kernel is loaded. Once the USBKey is on, the Rev_Verify_Mod will be verified and loaded. Then, the Rev_Verify_Mod reads the trust

metric base from the safe storage unit in USBKey. At last, the Rev_Verify_Mod sequentially reads the boot information of each phase and carries out the trust measurement and verification by comparing the current trust metrics with the premeasured values. If they are not equal, the system state is set as distrusted and the corresponding trusted boot failure handler is activated. If they are equal, the current system state is set as trusted and trusted boot continues to the next phase. When all system boot phases are completed successfully, the operating system is verified to be trusted and the trusted boot succeeds.

As shown in Figure 4, the steps are as follows:

(1) Insert USBKey and system is powered on. The BIOS boots first, and then the operating system is loaded and boots until the Init process is successfully loaded and user interface is active.

(2) USBKey is enabled and the Init process requires user inputs PIN code to proceed to USBKey authentication.

(3) If USBKey authentication succeeds, it is ready to be used to proceed to trusted boot. First, the legitimacy and integrity of the Rev_Verify_Mod are verified by USBKey by comparing the calculated results with the prior measurement values. If they are identical, the authentication is successful. Then the Rev_Verify_Mod is loaded by operating system kernel, and the control is handled over to the Rev_Verify_Mod module.

(4) The Rev_Verify_Mod module reads the prior measurement values of each boot stage from trust metric base inside USBKey, which contains BIOS metric, Grub Stage 1 metric, Grub Stage 1.5 metric, Grub Stage 2 metric, the metric of the Grub configuration file, the operating system kernel image metric, Init process metric, kernel module metrics, and daemon metrics.

(5) The Rev_Verify_Mod module reads the BIOS information and measures its integrity with USBKey. By comparing the calculated results with the records of the BIOS metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(6) The Rev_Verify_Mod module reads the Grub Stage 1 data and measures its integrity with USBKey. By comparing the calculated results with the records of the Grub Stage 1 metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(7) The Rev_Verify_Mod module reads the Grub Stage 1.5 data and measures its integrity with USBKey. By comparing the calculated results with the records of the Grub Stage 1.5 metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(8) The Rev_Verify_Mod module reads the Grub Stage 2 data and measures its integrity with USBKey. By comparing the calculated results with the records of the Grub Stage 2 metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(9) The Rev_Verify_Mod module reads the Grub configuration file and measures its integrity with USBKey. By comparing the calculated result with the records of the Grub configuration file metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(10) The Rev_Verify_Mod module reads the operating system kernel image file and measures its integrity with USBKey. By comparing the calculated result with the records of the operating system kernel image file metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(11) The Rev_Verify_Mod module reads the Init process file and measures its integrity with USBKey. By comparing the calculated result with the records of the Init process file metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(12) The Rev_Verify_Mod module reads the data of the kernel modules to be loaded and measures its integrity with USBKey. By comparing the calculated results with the records of the kernel modules metrics, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(13) The Rev_Verify_Mod module reads the data of the daemons to be loaded and measures its integrity with USBKey. By comparing the calculated results with the records of the daemons metrics, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(14) Operating system boot is complete. During the process of the above reverse integrity verifications, the chain of trust is established by loading order. If any step encounters a verification failure, the system trusted boot fails and the system state is set to be distrusted and the corresponding trusted boot

failure handler will be called which will normally suspend the system boot process or switch to the distrusted boot process. If all integrity measurements are successful in all steps, then the operating system is set to trusted status, and the trusted boot succeeds.

(15) The operating system trusted boot based on reverse integrity verification succeeds and system runs in a trusted computing environment.

3.4. Comparative Analysis. USBKey is featured with low cost, safe, portable, convenient, and other characteristics. Compared with TPM, it can provide similar security functions with higher flexibility at the same time. They both have secure data storage space for sensitive data, such as trust metrics, digital certificates, and keys. Reading and writing operations on the safe storage space can only be achieved through the COS inside the hardware, which prevents the user from directly reading secret data. The user keys inside the safe hardware cannot be exported, which eliminates the possibility of the replication of user digital certificate or identity information. A variety of cryptographic algorithms are performed inside the safe hardware with the CPU inside. Operations such as encryption, decryption, and signature operations are performed within USBKey to ensure that the key will not appear in the computer memory, so as to prevent the fact that the user key may be intercepted by attackers. Therefore, by applying the USBKey and the reverse integrity verification method on operation system trusted boot, the system integrity can be guaranteed, and better flexibility and convenience are achieved.

In existing USBKey-based security enhancements methods, the USBKey is primarily used for user authentication, such as USBKey-based two-factor authentication, and data encryption and decryption. Our work in this paper extends the application field of USBKey. By constructing prior measurement libraries and reverse integrity verification modules, the USBKey in our method can simulate TPM and support key functions in trusted computing as trusted boot, trusted storage, and remote attestation.

The main advantages of the proposed method include the following:

(1) In the operating system trusted boot method based on the reverse integrity verification, the system is guaranteed by verifying the integrity of data and executable files in different booting stages. The verifications are not sensitive to loading sequence and are performed stage by stage to ensure that the system is in a trusted status. Compared with TPM-based trusted boot, USBKey based trusted boot is more compatible with application environment and is more flexibility and easier to be used in real computing systems.

(2) With the operating system trusted boot method based on the reverse integrity verification, the operating system trusted boot can be achieved on endpoints without TPMs, which greatly enhances the applicable scope of trusted computing technology. At the same time, the procedure for trusted boot is simplified, and trust measurements and verifications can be performed at any phase of system boot process through the reverse attestation method.

Compared with TPM-based trusted boot, the main disadvantage for our method is that the system is unprotected before the Rev_Verify_Mod module is loaded, which makes it possible to bypass the reversed verification procedure and boot a distrusted system. One solution is to enable USBKey and implement USBKey-based reversed verification in BIOS boot phase to achieve the lower level protection. Another option is to encrypt the file system with USBKey, which only can be decrypted after the system is verified to be trusted by the Rev_Verify_Mod module.

4. A Scenario for Remote Attestation Based on the Reverse Integrity Verification Method

4.1. Related Works about Remote Attestation. Remote attestation is one of the core functions in trusted computing. Users are able to authenticate the identity of the target platform and measure and verify the integrity of the trusted computing platform with remote attestation by using TPMs or TCMs.

Remote platform authentication is one of the key mechanisms in trusted computing, whose purpose is to prove the identity of a remote entity by exchanging and verifying a series of certificates with TPM. In 2004, Brickell and other scholars proposed TPM-based direct anonymous attestation (DAA) program [13], which utilized zero-knowledge proof and group signature technology to prove the identity of the platform, but it was too complex to be implemented. Following DAA program, many new methods have been proposed to improve the efficiency and usability of DAA [14–18].

TCG defines a binary remote attestation protocol to attest the integrity of the remote platform with trusted hardware TPMs/TCMs. But it is argued for the overexposure of the configuration information about the hardware and software in local trusted computing platform. To overcome the flaw in binary remote attestation, Chen and other scholars firstly proposed a property based attestation protocol, PBA protocol [19]. Later, PBA protocol was further studied, and more remote attestation methods based on properties were proposed [20–24]. So far, property based remote attestation solution has currently been viewed as being the most valuable and prosperous for remote attestation, which overcomes the problems in binary remote attestation such as complexity, privacy, misuse of proof, and other defects.

In China, the Trusted Connection Architecture (TCA) [25] is proposed as a standard remote attestation protocol which is an improvement of TCG's Platform Trusted Service (PTS). The basic authentication model in TCA is shown in Figure 5.

Before establishing a trusted network connection, Access Requester (AR) and Access Controller (AC) must separately load PTS by calling the specific platform binding functions. Then the bidirectional user authentication protocol is performed, in which Policy Management (PM) acts as a Trusted Third Party (TTP). Platform A and Platform B collect

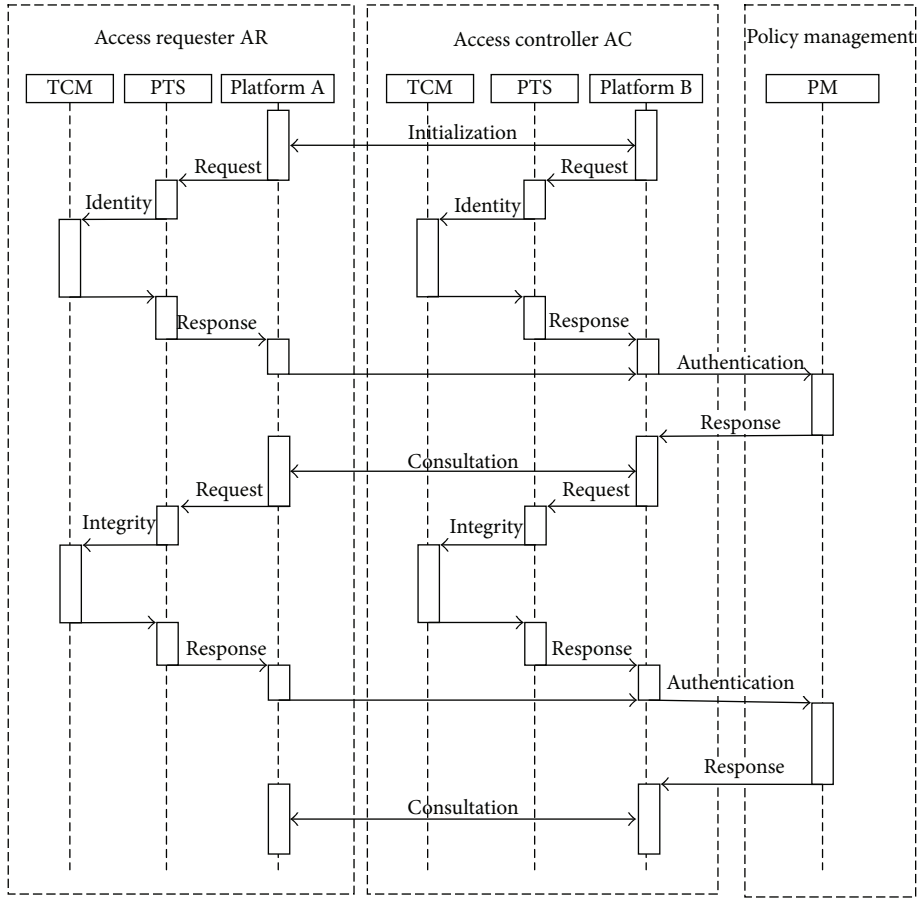


Figure 5: The basic authentication model in TCA.

integrity information with PTS protocol and then send the integrity information to PM. Finally the integrity of Platform A and Platform B is attested by PM.

4.2. A Scenario for Remote Attestation Based on the Reverse Integrity Verification Method in Hybrid Trusted Network. The remote attestation of the platforms relies on TPM chips; therefore, it is hard for PTS to be widely used in practical network environment since most of the servers and the terminal nodes are not equipped with TPM/TCM chips. The USBKey-based reverse integrity verification method in this paper can simulate TPM to measure trusted boot process and verify the trust state of the node. The system trust metrics and simulated PCR values are stored inside safe hardware with same data structures, which can be used to support remote attestation with other trusted nodes with TPM. Therefore, it is possible to deploy USBKey-based reverse integrity verification mechanism on nodes without TPMs and build a hybrid trusted network environment supporting remote attestation on all nodes.

In a hybrid trusted network environment, bidirectional remote attestation protocol between TPM-based trusted nodes and USBKey-based trusted nodes is designed as follows.

As shown in Figure 6, a new PTS protocol is designed with only a few simple modifications upon the original one to mask the differences between TPM/TCM and USBKey. A transparent layer is implemented to support both TPM/TCM and USBKey in remote attestation. Both PCR values generated by TPM/TCM and simulated PCR values generated by USBKey are collected as trust evidence. Thereby bidirectional remote attestation between TPM-based trusted nodes and USBKey-based trusted nodes can be achieved, which greatly reduces the cost of building an enterprise trusted network.

5. Conclusion

Aiming at the limitations in deploying and using TPMs in practical applications, we propose USBKey-based reverse integrity verification model which uses the widely used USBKey to establish chain of trust instead of TPM. The detailed design and implementation for this method are presented in Linux platform. The method supports the trusted boot of the Linux operating system. The PCR values are simulated to support the bidirectional remote attestation and trusted network connections between TPM-based trusted nodes and USBKey-based trusted nodes.

The proposed method greatly reduces the threshold for applying trusted computing technology. It has a wide

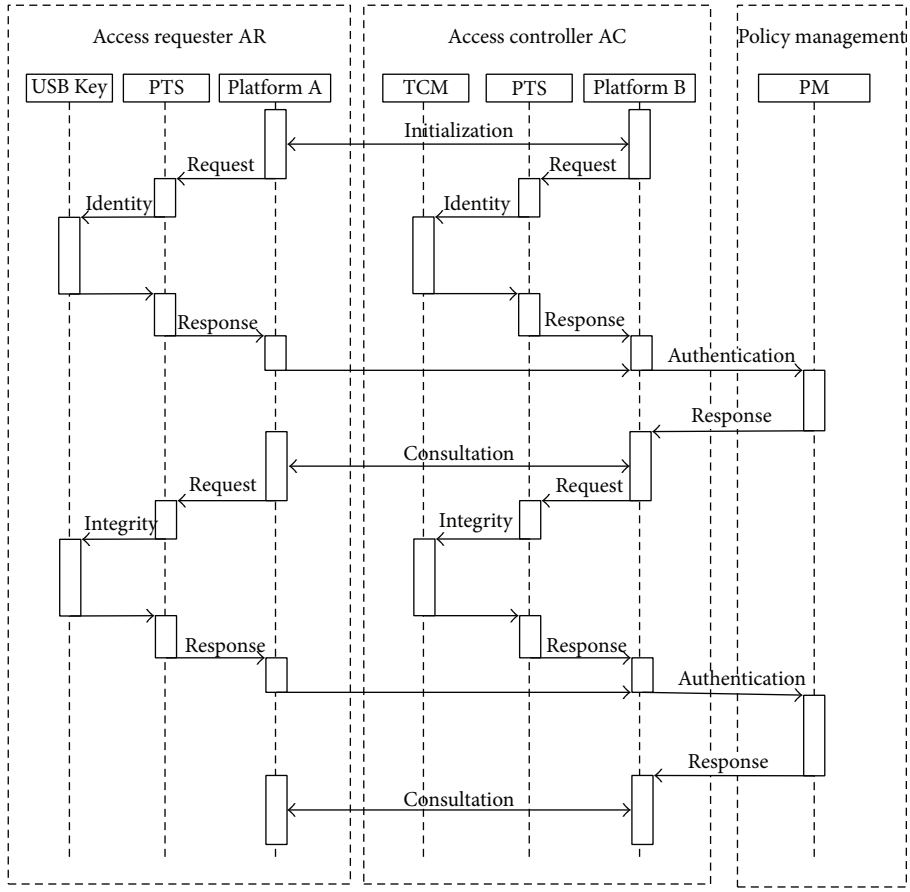


Figure 6: Remote attestation in hybrid trusted network.

prospective of applications and contributes a lot to the popularization of trusted computing technology in real enterprise environment.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant nos. 61303191 and 61502510), the HGJ Major Project of China under Grant no. 2015ZX01040301, and Foundation of Science and Technology on Information Assurance Laboratory (no. KJ-14-107).

References

[1] Trusted Computing Group, Trusted Platform Module Specification [EB/OL], http://www.trustedcomputinggroup.org/developers/trusted_platform_module/specifications.

[2] China TCM Unit, http://www.ztcia.com.

[3] X. Zhang and C. Shen, "A novel design of trusted platform control module," *Geomatics and Information Science of Wuhan University*, vol. 33, no. 10, pp. 1011–1014, 2008.

[4] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "Trusted execution environments on mobile devices," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*, pp. 1497–1498, Berlin, Germany, November 2013.

[5] R. S. Sujeen and S. Periasami, "Verifying trusted code execution using ARM trustzone," *International Journal of Computer Science and Network Security*, vol. 13, no. 10, pp. 41–46, 2013.

[6] T. Nyman, J. E. Ekberg, and N. Asokan, "Citizen electronic identities using TPM 2.0," in *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices (TrustED '14)*, pp. 37–48, Scottsdale, Ariz, USA, November 2014.

[7] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng, "Providing root of trust for ARM TrustZone using on-chip SRAM," in *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*, pp. 25–36, ACM, Scottsdale, Ariz, USA, November 2014.

[8] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proceedings of the Conference on Hot Topics in Cloud Computing (HotCloud '09)*, p. 3, June 2009.

[9] H. Banirostam, A. Hedayati, A. K. Zadeh, and E. Shamsinezhad, "A trust based approach for increasing security in cloud computing infrastructure," in *Proceedings of the 15th International Conference on Computer Modelling and Simulation (UKSim '13)*, pp. 717–721, IEEE, Cambridge, UK, April 2013.

[10] S. M. Habib, S. Ries, M. Mühlhäuser, and P. Varikkattu, "Towards a trust management system for cloud computing marketplaces: using CAIQ as a trust information source,"

Security and Communication Networks, vol. 7, no. 11, pp. 2185–2200, 2014.

[11] T. Müller, H. Spath, R. Mäckl, and F. C. Freiling, "Stark," in *Financial Cryptography and Data Security*, pp. 295–312, Springer, Berlin, Germany, 2013.

[12] A. S. Kushwaha, "A trusted bootstrapping scheme using USB key based on UEFI," *International Journal of Computer and Communication Engineering*, vol. 2, no. 5, pp. 543–546, 2013.

[13] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS '04)*, pp. 132–145, ACM, 2004.

[14] D. Bernhard, G. Fuchsbauer, E. Ghadafi, N. P. Smart, and B. Warinschi, "Anonymous attestation with user-controlled linkability," *International Journal of Information Security*, vol. 12, no. 3, pp. 219–249, 2013.

[15] L. Chen and J. Li, "Flexible and scalable digital signatures in TPM 2.0," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*, pp. 37–48, November 2013.

[16] G. Proudler, L. Chen, and C. Dalton, "Direct Anonymous Attestation (DAA) in more depth," in *Trusted Computing Platforms*, pp. 339–352, Springer International, Berlin, Germany, 2014.

[17] L. Yang, J. Ma, W. Lou, and Q. Jiang, "A delegation based cross trusted domain direct anonymous attestation scheme," *Computer Networks*, vol. 81, pp. 245–257, 2015.

[18] B. Smyth, M. D. Ryan, and L. Chen, "Formal analysis of privacy in Direct Anonymous Attestation schemes," *Science of Computer Programming*, vol. 111, no. 2, pp. 300–317, 2015.

[19] L. Chen, R. Landfermann, H. Löhr, M. Rohe, A.-R. Sadeghi, and C. Stüble, "A protocol for property-based attestation," in *Proceedings of the 1st ACM Workshop on Scalable Trusted Computing (STC '06)*, pp. 7–16, ACM, November 2006.

[20] L. Chen, H. Löhr, M. Manulis et al., "Property-based attestation without a trusted third party," in *Information Security*, pp. 31–46, Springer, Berlin, Germany, 2008.

[21] J. Li, Y. Li, Y. Hu, H. Wang, and W. Liu, "An improved protocol for property-based attestation," in *Proceedings of the 32nd Chinese Control Conference (CCC '13)*, pp. 6343–6348, Xi'an, China, July 2013.

[22] V. Varadharajan and U. Tupakula, "Counteracting security attacks in virtual machines in the cloud using property based attestation," *Journal of Network and Computer Applications*, vol. 40, no. 1, pp. 31–45, 2014.

[23] Y. Liang, K. E. Guo, and J. Li, "The remote attestation design based on the identity and attribute certificates," in *Proceedings of the 11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP '14)*, pp. 325–330, Chengdu, China, December 2014.

[24] X.-H. Yue and F. Zhou, "An efficient property-based attestation scheme with flexible revocation mechanisms," in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW '12)*, pp. 1223–1230, Shanghai, China, May 2012.

[25] S. Changxiang and Z. Yuelei, "Trusted connect architecture," Chinese Standard GB/29828-2013, 2014.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 1496714, 13 pages
http://dx.doi.org/10.1155/2016/1496714

Research Article

Data Sets Replicas Placements Strategy from Cost-Effective View in the Cloud

Xiuguo Wu

School of Management Science and Engineering, Shandong University of Finance and Economics, Jinan 250014, China

Correspondence should be addressed to Xiuguo Wu; xiuguosd@163.com

Received 29 December 2015; Accepted 24 April 2016

Academic Editor: Ligang He

Replication technology is commonly used to improve data availability and reduce data access latency in the cloud storage system by providing users with different replicas of the same service. Most current approaches largely focus on system performance improvement, neglecting management cost in deciding replicas number and their store places, which cause great financial burden for cloud users because the cost for replicas storage and consistency maintenance may lead to high overhead with the number of new replicas increased in a pay-as-you-go paradigm. In this paper, towards achieving the approximate minimum data sets management cost benchmark in a practical manner, we propose a replicas placements strategy from cost-effective view with the premise that system performance meets requirements. Firstly, we design data sets management cost models, including storage cost and transfer cost. Secondly, we use the access frequency and the average response time to decide which data set should be replicated. Then, the method of calculating replicas' number and their store places with minimum management cost is proposed based on location problem graph. Both the theoretical analysis and simulations have shown that the proposed strategy offers the benefits of lower management cost with fewer replicas.

1. Introduction

Today, several cloud providers offer storage as a service, such as Amazon S3 [1], Google Cloud Storage (GCS) [2], and Microsoft Azure [3]. All of these services provide storage in several data centers distributed around the world. Clients can store and retrieve data sets without buying and maintaining their own expensive IT (Information Technology) infrastructures. Ideally, CSPs (Cloud Service Providers) should be able to provide low-latency service to their clients by leveraging the distributed locations for storage offered by these services. However, today's Internet still cannot guarantee quality of services and potential congestions can result in prolonged delays. Replication technology has been commonly used to minimize the communication latency by bringing the copies of data sets close to the clients [4]. Moreover, they also provide data availability, increased fault tolerance, improved scalability, and reduced response time and bandwidth consumption. Amjad et al. [5] have presented various dynamic replication strategies.

Compared with the definitions of conventional computing paradigms such as cluster [6], grid [7], and peer-to-peer (p2p) [8], "economics" is a noticeable keyword in cloud computing which has been neglected in data sets replicas placements. "Economics" denotes that cloud computing adopts a pay-as-you-go model, where clients are charged for consuming cloud services such as computing, storage, and network services like conventional utilities in everyday life (e.g., water, electricity, gas, and telephony) [9]. However, most current replicas approaches largely focus on improving reliability and availability [10, 11] by providing users with different replicas of the same service, ignoring the management cost spending on replicas, which cause great financial burden (storage cost, transfer cost, etc.) not only for cloud users, but also for CSPs.

It is obvious that the client access latency can be reduced with the number of replicas increased. And every client demands to access its data set from a replica that is as close as possible in order to minimize its delay. However, there are at least two challenges while replicating all data sets to all data centers can ensure low-latency access [12]. First, the system can not offer unlimited storage resources, and that approach is costly and may be inefficient for the extra storage resource consumption. Second, the problem is more complicated when the data set may be updated, and the more the replicas

are in the system, the higher the update cost will be. Thus, data set replicas need to be carefully placed to avoid unnecessary expense. And the replicas number and their store placements may have a profound impact on the optimal replicas distribution in a balance way from cost-effective view. The resulting tradeoff between the number of replicas and the data set delay maps precisely to the replicas placement problem.

In practice, CSPs supply a pool of resources, such as hardware (storage, network), development platforms, and service at the expense of cost. And data set storage and transfer costs are the two most important components in data management, which are caused by storage resource and bandwidth consumption, respectively. Also, with the number of new replicas increased, the transfer cost will be declined because the data set can transfer more effectively; but the storage cost is getting bigger because of new replicas' existence. That is to say, too many replicas in the cloud may lead to high storage cost, and the increased storage cost for replicas may be greater than the reduced transfer cost, if there is no suitable replicas placements strategy. Therefore, it is urgent to find a balance selectively to replicate the popular data or not.

Based on the analysis above, there are at least three important issues that must be solved in order to achieve the minimum-cost data set replicas placements scheme: (1) whether or not to create a replica in cloud computing environment; (2) how many data set replicas should be created in the cloud; (3) where the new replicas should be placed to meet the system task successful execution rate and bandwidth consumption requirements.

Therefore, in this paper, towards achieving the minimum-cost replicas distribution benchmark in a practical manner, we propose a replicas placements strategy model, including the way to identify the necessity of creating replica, and design an algorithm for replicas placements that can easily reduce the total cost in the cloud.

The main contributions of this paper include (1) proposing data sets management cost models, involving storage cost and transfer cost; (2) presenting a novel global data set replicas placements strategy from cost-effective view named MCRP, which is an approximate minimum-cost solution; (3) evaluating replicas placements algorithms using analysis and simulations.

The remainder of this paper is organized as follows: Section 2 presents related works in data set replicas placement and management cost. Then, in Section 3, data sets cost models are proposed, and a replicas scarce resource test algorithm is shown in Section 4. Section 5 describes the data sets replicas number and store places from cost-effective view based on Steiner Graph. Section 6 addresses the simulation environments, parameters setup, and performance evaluations of proposed replicas solutions. Finally, conclusions and future works are given in Section 7.

2. Related Works

We will present in this section the background related to data sets replicas placements and management cost models in the cloud.

2.1. Data Sets Placements Strategies. Data sets replication is considered to be an important technique used in cloud computing environment to speed up data access, reduce bandwidth consumption and user's waiting time, and increase data availability [13]. Data sets replicas placement is the problem of placing duplicate copies of data set in the most appropriate node in the cloud, which can be logically divided into two stages, namely, replication decision and replicas placements. The replication decision stage decides whether to create the replica. If the decision is not to replicate, the data set will be read remotely. The second stage is to select the best sites to store the new replicas. There are two types of replicas placements techniques: centralized and distributed. Distributed replicas placements can be further classified according to different implementation: free-scale topology [14, 15]; graph topology [16]; multitier architecture [17]; hierarchical architecture [18, 19]; peer-peer architecture [20]; tree architecture [21]; and so on. As a typical centralized replication placements technology, Andronikou et al. have proposed a dynamic QoS-aware data replication technique which is based on data importance in [22]. Kalpakis et al. considered the minimum cost of servicing read and write requests in a distributed system; however it is in a tree network [23].

Conclusions as a result, the above-mentioned replication technologies have not involved data set storage and transfer cost, which are the most important elements for the clients in deciding whether or not to use cloud storage system, especially for small business. Therefore, we will consider the data sets management cost as a basis for replicas placements in order to minimize the storage and transfer costs on the premise that system performance satisfies data set availability requirements in this paper.

2.2. Data Set Management Cost. In a pay-as-you-go paradigm, all the resources in the cloud carry certain costs, so the more the replicas the more we have to pay for the corresponding resources used. Some of them may often be reused while the others may not be. So, once we decide to create a replica, we need to evaluate its access frequency as well as management cost, especially when large data sets—or "big data"—are usually common in the cloud. In [23], toward practically achieving the minimum data set storage cost in the cloud, a runtime local-optimization based storage strategy has been developed. The strategy is based on the enhanced linear CTT-SP algorithm used for the minimum-cost benchmarking. Theoretical analysis and random simulations have shown its validity and reliability. Auction protocol is used by Bell et al. to select the best replica of a data file, where the total cost is computed as the sum of file transfer cost and estimated queuing time for all jobs in the queue [24].

Base on the analysis, it is very necessary to design the data sets replicas placements strategy from cost-effective view. And this research is very significant for business, especially for small businesses, which usually use big data on cloud computing platforms.

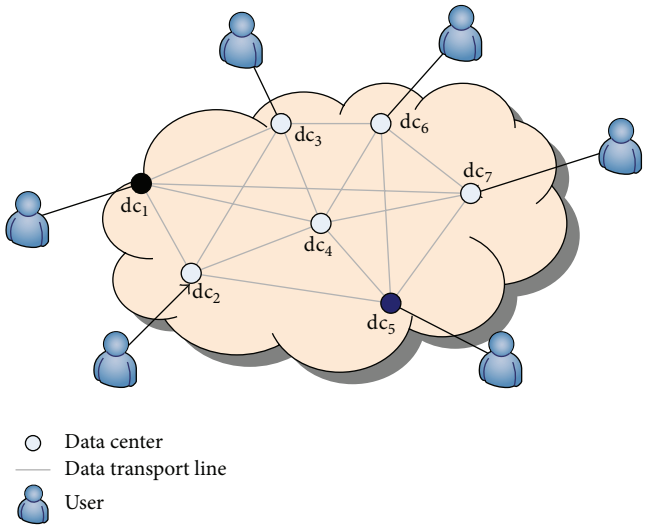


Figure 1: Architecture of cloud environment.

3. Data Sets Cost Models in the Cloud

In this section, we will first present some concepts in cloud environment; then we propose storage cost model and single transfer cost model, respectively. At last, we present data set management cost model in the cloud.

3.1. Some Concepts in the Cloud. In cloud storage system, there are some distributed data centers in cloud environment for data sets storage. And each data center has some properties, such as storage capacity, CPU speed, and network bandwidth, and read/write speed. Similarly, different configurations of data center lead to different quality of service (QoS).

Definition 1 (cloud computing environment, CCE). Cloud computing environment (CCE) can be regarded as a set of distributed data centers, written as CCE $= \bigcup_{i=1,2,\ldots,|DC|}\{dc_i\}$, where dc_i denotes the ith data center.

Figure 1 describes basic architecture of cloud computing environment constituted by seven data centers.

Definition 2 (data center, DC). In CCE, each data center dc_i can be described as a 5-tuple: $(dc_i, s_i, vs_i, sp_i, tp_i)$, where dc_i is the identifier of data center, which is a unique identification in CCE; s_i is the total space of data center dc_i, whose unit is TB; vs_i is the size of vacant space on data center, which means the extra storage capacity of dc_i; sp_i means the storage price of data, determined by the service provider; and tp_i is transfer cost ratio with each unit size data set.

Definition 3 (data set, d). Data set d_m in CCE can be described as a 3-tuple: (d_m, s, p), where d_m is the identifier of data set, and it is unique in the whole cloud environment; s is the size of data set; and p is its store place, $p \in$ DC.

In the following section, we assume that the architecture of CCE and data set d_m are relatively fixed during a period of time for simplicity.

3.2. Data Sets Storage Cost Model. In a commercial cloud computing environment, service providers have their cost models to charge users. For example, Amazon cloud service's prices are as follows: \$0.15 per gigabyte per month for the storage resource [1]. Storage cost depends on parameters such as the CSP's price policy, the size of the data set (original data set and inserted data set), and the storage time.

Definition 4 (storage cost, c_s). Data set d_m's storage cost is a function of its data size $d_m \cdot s$, storage time t, and its deployed data center dc_i's storage cost ratio sp_i, and can be represented as follows: $c_s = sp_i \times d_m \cdot s \times t$.

That is to say, the total storage cost is the CSP's storage cost ratio function multiplied by the size of the data set and its storage time, for example, using Amazon S3 for storage pricing and considering that 0.5 T (512 G) data set has been stored for 6 months. The storage cost is \$0.15 $*$ 512 $*$ 6 $=$ \$460.8.

3.3. Data Sets Transfer Cost Model. In the cloud, the data sets transfers are absolutely necessary once a request arrives, in which process the transfer cost will be generated inevitably for the reason for network consumption. In this model, the input data sets transfers are free, whereas output transfer cost varies with respect to data set volume and the CSP's atomic transfer cost ratio function.

Definition 5 (transfer cost, c_t). Data set d_m's transfer cost is the product of its data sets transfer time t_t and data center dc_i's atomic transfer cost tp_i, which can be described as follows: $c_t = tp_i \times t_t$.

It is noted that the transfer time t_t depends heavily on the data set size and network bandwidth. And in practice, the bandwidth may fluctuate from time to time according to peak and off-peak data access time. In this paper, we simplify the problem and regard the bandwidth as a static value, ignoring the volatility over time. For example, for a 10 G data set, the single transfer cost is \$0.12 $*$ 10 $=$ \$1.2, if the transfer cost ratio is \$0.12 per GB data set.

3.4. Data Sets Management Cost Model. In this paper, we facilitate a data set d_m's management cost during a period depending on several parameters: the size of the data sets $d_m \cdot s$, the time t, and the requests times n during t. And d_m's total management cost can be defined as follows.

Definition 6 (data set d_m's total management cost, tc). Data set d_m's total management cost during a period of t is the sum of its storage cost c_s and transfer cost c_t: $tc = c_s + n \times c_t = sp_i \times d_m \cdot s \times t + tp_i \times t_t \times n$, where n is the requested times.

Let us introduce a simple example: a 500 G data set is stored in the cloud for a month, and the storage cost is \$0.1 per GB and per month. The transfer cost is \$0.12 per GB data set, and the number of requests is 10. Then, storage cost is \$50, and transfer cost is \$600, for a total of \$650 in a month.

Input: data set d_m, ε_{af}, ε_{rt}, t;
Output: if d_m is a replica scarce resource, return true; else return false;
(01) count the data set d_m's requested access times n using logs files;
(02) set $af_{d_m} = n/t$;
(03) sum the data set d_m's response time rt_i using logs files;
(04) set $art_{d_m} = (\sum_{k=1}^{i} rt_k)/n$;
(05) set $rsr_{d_m} = art_{d_m}/d_m \cdot s$;
(06) if $((af_{d_m} > \varepsilon_{af})$ and $(rsr_{d_m} > \varepsilon_{rt}))$
(07) return true;
(08) else
(09) return false;
(10) End.

Algorithm 1: Replica scarce resource testing.

4. Replicas Scarce Resource Model in the Cloud

In this section, we will present the replicas scarce resource model in order to determine whether or not to create a new replica in the cloud.

4.1. Replicas Scarce Resource. There are many data sets stored on the data center in the cloud environment. And it is not necessary to replicate the data set on all the data centers. It is intelligent to replicate the popular data sets with high user frequencies for reducing data set transfer delay. In this way, we will define replica scarce degree as a criterion of adding replicas.

Definition 7 (access frequency, af). For a data set d_m, its access frequency is the requested access times per unit of time and can be represented as follows: $af = \sum_{i=0}^{n} times_i/n$, where $times_i$ indicates the number of accesses to the replicas on data center dc_i of unit time interval and n is the total number of replicas.

If a data set d_m's access frequency is greater than a preset threshold ε_{af}, then data set d_m is hot data.

Definition 8 (response time, rt). Response time rt is the time that elapses when a service requests a data set until the user receives the complete data set.

It is obvious that average response time can be calculated starting from the initial time when the request is submitted till the final response if received with the image from the target node.

Definition 9 (average response time, art). Average response time art is the ratio of total response time and the requested times per unit of time and can be represented as follows: $art = (\sum_{k=1}^{m} rt_k)/m$, where m is the requested times during a period of time t.

Average response time (art) is a basic parameter to determine the replicas numbers and stored places for the reason that the awt can be reduced by placing replicas on data centers. However, awt is not the only valid parameter to create replicas. The reason is the average response time depends on a number of factors, such as bandwidth and data set size al. And the bigger the data set size, the longer the average response time. In this way, we will present replica scarce resource by introducing data set size.

Definition 10 (replica scarce resource, rsr). A data set d_m is a replica scarce resource if the ratio of a data set d_m's average response time art_{d_m} and its size are less than a preset threshold ε_{rt}, which can be described as $art_{d_m}/d_m \cdot s > \varepsilon_{rt}$.

To sum up, it is necessary to create replicas for replica scarce resource. And there are two important factors to be considered before creating new replica: (1) longer average response time and (2) higher requests frequency.

For those data sets with low requested frequency, and those with high requested frequency but short response time, there is no need to place replicas from cost-effective view. Algorithm 1 describes whether it is necessary to create replicas for data set d_m.

Algorithm 1 presents the way to determine the possibility of creating replicas. And its time complexity is $O(n)$, for the reason that line (04) sums up all the response time n times.

4.2. Replica Placements from Cost-Effective View. Once the decision to create replicas has been made, the most urgent problem needed to solve is where to place it. In this subsection, we will present a replica placement algorithm from cost-effective view.

It is obvious that the replica's candidate store places are not unique, but a set of data centers. Then the most economic way is to select the data centers with lower storage cost and transfer cost to other data centers on the basis of shorter average response time. And Algorithm 2 presents the suitable stored places choosing schema by comparing the data set management cost during a period of time t.

In Algorithm 2, line (01) defines the total cost cc as a largest value, which will be modified immediately after the first cycle. And the function min() in line (10) and line (25) will return the smaller response time and transfer cost, respectively. Line (14) is mainly used to compare average

Input: data set d_m, data centers set DC = $\{dc_0, dc_1, dc_2, \ldots, dc_n\}$, dc_0 stores the primitive data set d_m, testing time t; pre-set average response time ε_{rt};
Output: data center dc_m with lowest cost;
(01) set cc = MAX_COST; //Assuming the cost is largest
(02) set $m = 0$; //Initialize return data center index
(03) for each data center dc_j (except data center dc_0)
(04) begin
(05) //Assuming the replica stores on dc_j;
(06) set $rs_j = 0$; //record total response time
(07) set $n_j = 0$; //record access times;
(08) for data center dc_k (except dc_0 and dc_j)
(09) begin
(10) set $rs_j = rs_j + n_k \times \min(rs_{k\_dc_0}, rs_{k\_dc_j})$;
(11) set $n_j = n_j + n_k$;
(12) end
(13) set $ars_j = rs_j / n_j$;
(14) if ($ars_j > \varepsilon_{rt}$)
(15) continue;
(16) else
(17) begin
(18) calculate the storage cost sc_j using Definition 4;
(19) set tc = 0; //transfer cost is initialized to zero
(20) set $n_j = 0$;
(21) for data center dc_k (except dc_0 and dc_j)
(22) begin
(23) calculate transfer cost $c_{tdc_k - dc_j}$ using Definition 5;
(24) calculate transfer cost $c_{tdc_k - dc_0}$ using Definition 5;
(25) set $tc = tc + n_k \times \min(c_{tdc_k - dc_j}, c_{tdc_k - dc_0})$;
(26) set $n_j = n_j + n_k$;
(27) end
(28) if $tc < cc$
(29) $m = j$;
(30) end
(31) end
(32) return m.

Algorithm 2: Select replica's economic stored placements.

response time before and after a replica is stored on data center dc_j. And the process will automatically break and turn into next cycle once average response time is still greater than preset threshold ε_{rt}.

Here, we will analyze the time complexity. Suppose there are n data centers, and in line (08), the loop times for transfer cost are $(n - 1)$, the same in line (21). So, the total time complexity is $O(n^2)$.

5. Data Sets Replicas Placements from Cost-Effective View

In the previous sections, we have tentatively placed one replica on a data center and obtained high system performance with lowest data sets management cost. However, replicas number and their storage places are still urgent problems to be solved from cost-effective view in practice. In this section, we will present an approximate minimum-cost replicas placements algorithm based on location problem (LP).

5.1. Minimum-Cost Replicas Placements Model. In order to formulate the minimum-cost replicas placements problem, we make the following assumptions: (1) The cloud computing environment is a customer-to-server system, in which the data sets themselves travel to the facilities to be served. On the other hand, the user requests the data set for further analysis, not a result by computing or querying from the data set. (2) Each data center represents a candidate replica location as well as a data set demand point, for the reason that client requests data via data center. (3) Only one replica may be located per data center. (4) The replicas service is uncapacitated; that is, they may serve an unlimited amount of data sets requests.

In its simplest form, the minimum-cost replicas placements problem is as follows: given a set of data centers, which represent demand points as well as candidate replicas placements, and a set of connections between each pair of data centers. Each connection has a transport cost per unit data set and each data center is associated with a charge for data set storage. Also, all demands must be routed over

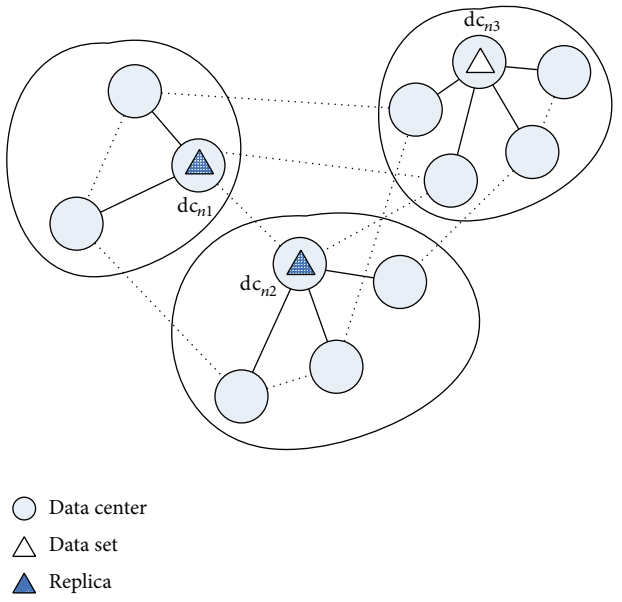


Figure 2: Data set access domain with minimum management cost.

the connection to the nearest replica. The problem is to find the set of replicas placements that minimize the total management cost: the sum of data sets storage and transport cost.

Ideally, the optimal minimum-cost replicas placement is shown in Figure 2, where each replica is stored in a domain D, and the rest of the data centers retrieve data set from it. In this way, the transfer lines and the data centers constitute a tree.

Next, we model the minimum-cost replicas placements problem. Conventionally, a cloud environment is represented by a graph where two nodes have an edge if and only if two corresponding nodes can communicate with each other. In order to describe such a circumstance, we will transform them into a graph G. The transformation rules are as follows:

(1) Each data center dc_i is mapped to a node dc_i in the graph, and all nods constitute the node set V.

(2) Each connection line between two data centers should be classified according to the type of network and used for implementation: (i) lines in one domain from a node with a replica to others can be transformed into edges with weight 0, respectively; (ii) lines in one domain between nodes without replica can be transformed into edges, respectively, and their weight is minimum transfer cost between corresponding data centers; (iii) lines cross-domain can be transformed into edges between corresponding nodes, and its weight is the minimum transfer cost.

(3) The storage cost of each data center should be mapped to the first property of corresponding node.

(4) The product of access frequency and time of period t is mapped to second property of corresponding node.

In order to describe the nodes, we define a 3-tuple (dc_x, c_x, p_x) representing the identifier, storage cost, and transfer times, respectively.

In this way, we wish to find optimal locations at which we place replicas to serve a given set of n clients; we are also given a set of locations at which replicas may be stored, where store replica at data center incurs a cost of $f_1(v_i)$, and each client j must be assigned to one replica (or source data), thereby incurring a cost of c_{ij}, proportional to the distance between data centers dc_i and dc_j; the objective is to find a solution of minimum total cost.

The minimum-cost replicas placements problem is defined as follows.

Definition 11 (minimum-cost replicas placements, MCRP). Given a connected undirected and weighted graph $G = (V, E, w, f_1, f_2)$, (1) V is the set of nodes; (2) E is the set of edges; (3) w is a function $w : e(v_i, v_j) \to R^+$; (4) f_1 is a function: $v_i \to R^+$, $v_i \in V$, and $f_1(v_i)$ is a nonnegative real value; (5) f_2 is a function $V \to Z^+ : v_i \to Z^+$, and $f_2(v_i)$ is a nonnegative integer value associated with each node. The goal is to divide V into two subsets V_1 and V_2, and V_1 is nodes set with replicas, while V_2 is nodes set without replicas and any $v_i \in V_2$ need to request data set from ($v_j \in V_2$). The task is to construct a forest constituted by trees. Each subtree consists of a source node (with replica) and multiple destination nodes such that the sum of transfer cost and storage cost is minimized. That is, the value from each node v_i to its root multiplied by $f_2(v)$ is minimized, represented as

$$\text{Minimize}\left(\sum_{v_i \in V_1(T)} f_1(v_i)\right) + \sum_{v_i \in V_2(T)} \sum d(T, v_i) f_2(v_i). \quad (1)$$

Several aspects of this formulation are worth noting. First, we observe that if we set the transfer cost to the same, then the result is simple, which is an alternate formulation of the uncapacitated facility location problem (UFLP) in which link additions are disallowed. Thus, the UFLP is a special case of the UFLNDP in which link additions are disallowed. Since the UFLP is NP-hard (in the parlance of computational complexity) [25, 26], so is the more general MCRP. Therefore, we can conclude the following theorem.

Theorem 12. *MCRP is NP-hard.*

In the original problem, we need to decide d_m's replicas number and their places to put. Once we select a data center dc_k, then the sum of the rest of the data centers transfer cost and the storage cost must be minimal compared to selecting other data centers. In the same way, the question that how many replicas need to be placed in the cloud platform has turned into how to select a subset of nodes in graph G that can minimize the sum of nodes weight and edges weight.

Then, it can be regarded as a UFLP using mapping rules shown in Table 1.

Note that the storage cost for the node and cost of the edges should be expressed in comparable units; for example, the storage cost for each node can be expressed in dollars for

Table 1: Mapping rules from minimum-cost replica placements strategy to UFLP.

Index	Element in minimum-cost replica placements strategy	Mapping	Element in UFLP
(1)	The transfer cost ratio between data centers c_t	→	$w(v_i, v_j)$
(2)	The storage cost of each data center c_s	→	$f_1(v_i)$
(3)	The user requested access times	→	$f_2(v_i)$
(4)	$\min\left(\text{Cost}_{\text{storage}}(\text{dc}_k) + \sum \text{Cost}_{\text{transfer}}\right)$	→	$\min(\text{Cost}(T))$

Input: Graph $G = (V, E, w, f_1, f_2)$ with edge-weighed and node-weighted;
Output: Graph $G' = (V', E', w')$ with edge weighed;
(01) Initialize an edge-weighed graph G'', by setting $V'' = V$, and $w'' = w$;
(02) For each $(v_i, v_j) \in E$, do
(03) Assign the weight of this edge as
(04) $w''(v_i'', v_j'') = w(v_i, v_j) + \max(f_1(v_i) + f_2(v_j), f_2(v_i) + f_1(v_j), f_2(v_i) + f_2(v_j))$;
(05) EndFor
(06) Initialize an edge-weighed graph G' by setting $V' = V''$, and $f_1' = f_1''$;
(07) For each $v_i \in V'$, do
(08) Assign the weight of edge from v_i to v_j
(09) $w''(v_i'', v_j'') = \text{Dijkstra}(v_i, v_j)$;
(10) Output G'.

Algorithm 3: Construct graph with edge weighted.

each replica, while the cost of each edge can be represented in dollars per request.

Theorem 13. *An optimal solution to the MCRP consists of p replicas and $|N|$-p connections, where N is the total number of data centers.*

This property quantifies our intuition about the tradeoff between constructing facilities and links; that is, as we build more facilities, fewer links are needed. The property also has implications in the identification of polynomial solvable cases, as has been discussed in [27].

In this way, the minimum-cost replicas placements problem is NP-hard. Therefore, no polynomial time algorithms of solving the problem are likely to exist for minimum-cost replica placements. Hence, it is of practical importance to obtain approximation methods whose costs are close to optimal.

5.2. Approximate Algorithm for Replicas Placements with Minimum Management Cost. In this subsection, we introduce an approximate algorithm for MCRP. The idea is first decomposing the transfer ratio to edge weight and then finding the candidate replicas placements data centers using graph G. Finally, the approximate solution for the graph G gives an approximate solution for the original problem.

5.2.1. Transformation from Edge and Node-Weighted Graph to Edge-Weighted Graph. First, we will construct a graph G and then move the user access frequency and storage cost to the edge as weight, constructing an edge-weighted graph. The basic idea can be summarized as follows: decomposing the storage cost on adjacent edges according to degree of data centers. Algorithm 3 describes the setting of edge weight.

In Algorithm 3, function Dijkstra(v_i, v_j) returns a minimum transfer cost from v_i to v_j. And the time complexity of Dijkstra is $O(n^2)$; we know that the complexity of Algorithm 3 is $O(n^3)$ from line (07).

5.2.2. Approximate Algorithm for MCRP. Next, we will propose approximate minimum management cost replicas placement algorithms based on the graph G'. The basic idea is to obtain possible replicas placements according to the minimum spanning tree, as is shown in Algorithm 4.

In Algorithm 4, function $\deg(v_i)$ means the number of edges linked to node v_i. And Algorithm 4 requires the number of nodes greater than two; if not, the nodes with minimum storage cost are the better ones.

Here, we will analyze the time complexity of Algorithm 4. A simple implementation using an adjacency matrix graph representation and searching an array of weights to find the minimum weight edge to add requires $O(|V|^2)$ running time. Kruskal Algorithm is greedy algorithm that runs in polynomial time, whose time complexity is $O(n^2)$; n is the number of vertexes. In the other steps, such as in line (03), its time complexity is $O(n^2)$. So, the total time complexity of MCRP is $O(n^2)$. A detailed example will be shown in Section 6.2.

The data center that deployed the original data set is responsible for the data set and its replicas' management, including when to create replicas and where to place them. With the data set requests increased (e.g., the number of requests amounts to 5000 in a month), the data set replicas placements algorithm with minimum management cost data

Input: Graph $G' = (V', E', w', f_1', f_2')$ with edge-nodes weighted as (dc_i, c_i);
Output: The nodes set for replicas placements.
(01) Generate a minimum-cost spanning tree from G' using Kruskal Algorithm;
(02) v_i = node with maximum degree;
(03) While $\deg(v_i) \geq 2$ do
(04) Begin
(05) v_i = node with maximum degree;
(06) Print v_i;
(07) For $v_j \in V$ and $i \neq j$ do
(08) If $e(v_i, v_j) \in E$
(09) Begin
(10) Delete $e(v_i, v_j)$;
(11) If $\deg(v_j) = 0$ Delete v_j;
(12) EndIF
(13) Delete v_i;
(14) v_i = node with maximum degree;
(15) EndWhile
(16) End.

Algorithm 4: Replicas placements with approximate minimum management cost.

center will start up, and the data set replicas will be transferred to other data centers according to the computation results. Also, a replicas distribution table including some important information such as original data set and its replicas' location will be placed, and its size, most recent update time, and so forth al, will send to each data center, in order to let the others know where the data sets are placed. In this model, a user that connected the data center accesses a data set as follows: First he/she tries to locate the data set replica locally. If the object replica is not present, he/she goes to check the data placement directory residing on each data center, which stored a data set replicas distribution structure. After that, the user's request goes to the nearest data center and then will transfer the data set to user via near data center.

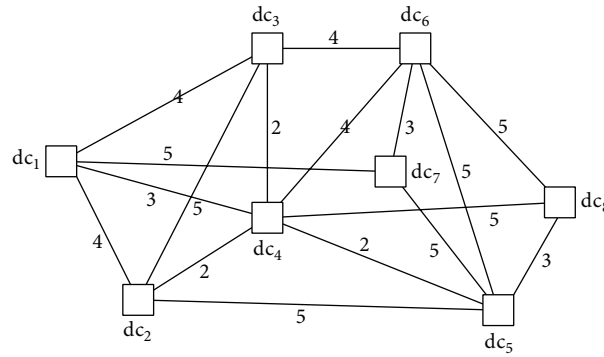


Figure 3: Architecture of cloud environment.

6. Analysis and Evaluation

In this section, we first present the experimental setups and then discuss the tradeoff between storage cost and transfer cost. Next, we describe the whole procedures of MCRP approximate algorithm using an example step by step. At last, we compare the result among different circumstances to demonstrate how our replicas placements strategy works.

6.1. Experimental Setup. The experiments were conducted on a cloud computing simulation environment built on the computing facilitates at Network & Information Security Lab, Shandong University of Finance and Economics (SDUFE), China, which is constructed based on SwinDeW [28] and SwinDeW-G [29–31]. The cloud system contains 10 data centers (servers) and 50 high-end PCs (clients), where we install VMWare (http://www.vmware.com), so that it can offer unified computing and storage resources.

Figure 3 describes the simulation architecture model of cloud storage platform, and the prices of cloud services follow the well-known Amazon's cost model: (1) \$0.15 per gigabyte per month for storage; (2) \$0.1 per gigabyte for data sets transfer process.

Table 2: Data sets access configurations.

Data centers	dc_1	dc_2	dc_3	dc_4	dc_5	dc_6	dc_7	dc_8
Users number m_i (×100)	1	0.6	0.8	1.1	0.6	0.7	0.9	1
Access frequency π (×10)	2	3	3	5	4	6	5	4

And in the analysis, we observe and study the running conditions for a period of one month. The usage frequency is according to Poisson distribution. Table 2 describes the data centers users' number m, access frequency π, and so forth.

6.2. Tradeoff between Storage and Transfer Costs. We define the cost of a solution in MCRP as the sum of storage cost and transfer cost. In order to compare the total costs with different replicas numbers, we have computed the storage and transfer costs, respectively. The result is shown in Figure 4. Every point on the black thick line in this diagram represents a value of sum cost.

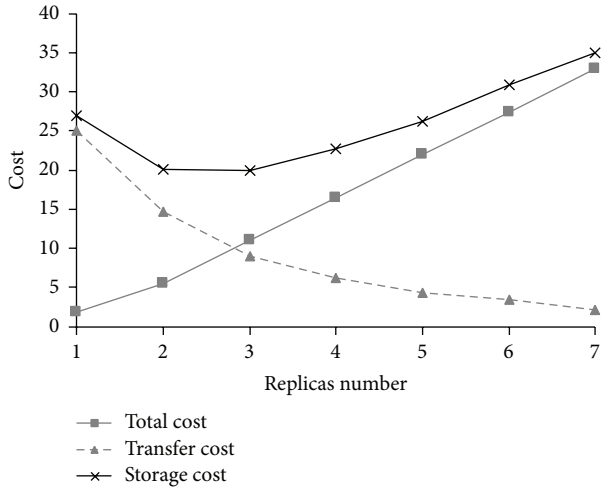


Figure 4: Tradeoff between storage and transfer costs.

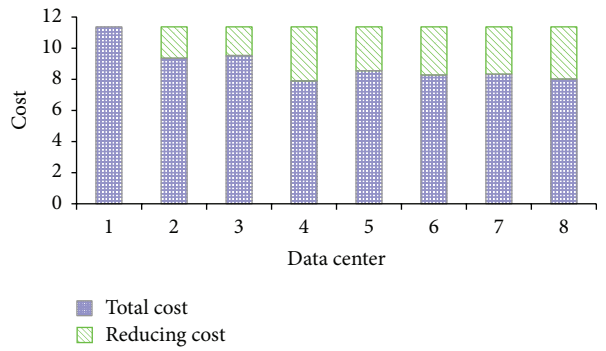


Figure 5: Reduced cost comparisons of data centers.

From Figure 4, we can see that, without replica, the storage cost is less than transfer cost for the reason that many data sets are transported in the system. However, with replicas number increased, the storage cost is gradually getting bigger for the reason that more replicas need to be stored on data centers, while the transfer cost becomes smaller for the nearby replicas access. Furthermore, the total cost has a minimum value with three replicas, which means the optimal solution.

Similarly, Figure 5 describes the reduced cost with one replica. And the green part is the value that can be saved. Also, it is obvious that the cost will be reduced with the replicas added. Also, we can see that the data center dc_4 is the replica storage place with maximum saved cost.

6.3. Approximate Algorithm for MCRP. In this section, we will analyze the MCRP algorithms proposed in Section 5.

First, we need to generate a minimum-cost spanning tree from Figure 3 using Kruskal Algorithm. And Figure 6 shows the result, including seven edges.

Then, we will select the node with maximum linked edges, for example, dc_4, and delete its adjacent edges. Also, if the degree of adjacent node is zero, then remove it at the same time, for example, nodes dc_1 and dc_2. Figure 7 shows the

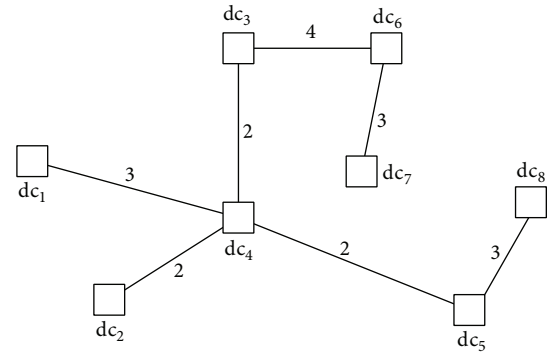


Figure 6: Minimum-cost spanning tree.

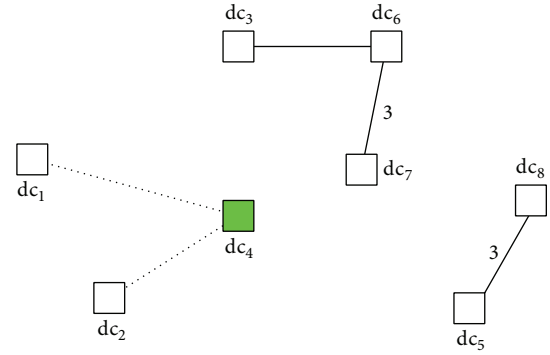


Figure 7: Result after first deletion.

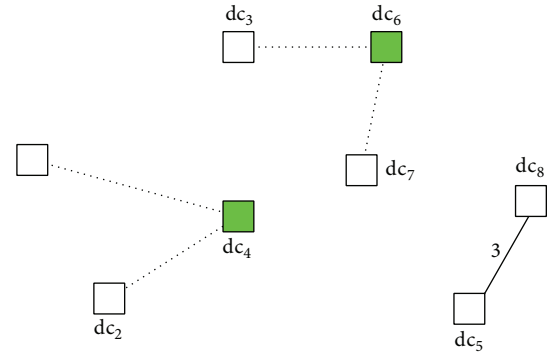


Figure 8: Result after second deletion.

result after first deletion. And in this step, dc_4 is the first selected replica store location.

It is obvious that there still exist nodes with degree greater than two, for example, dc_6, and then the node should be deleted and also its linked edges. In this way, dc_6 is the second selected replica store location. Figure 8 shows the result after second deletion.

In this case, the maximum degree of all nodes is only one, and then the node with small storage cost value is the suitable place to store replica, for the reason that they have the same transfer cost despite where the replica store place is. Figure 9 shows the result of replica store place, that is, a node set of $\{dc_4, dc_6, dc_8\}$.

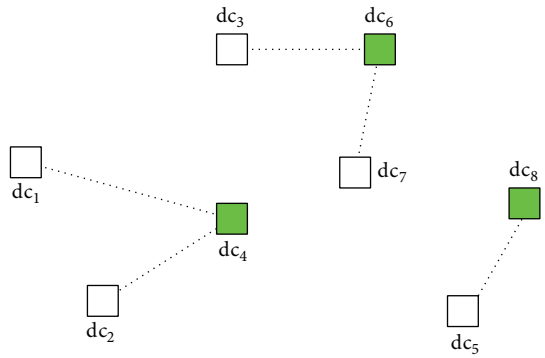


Figure 9: Result of replicas store places $\{dc_4, dc_6, dc_8\}$.

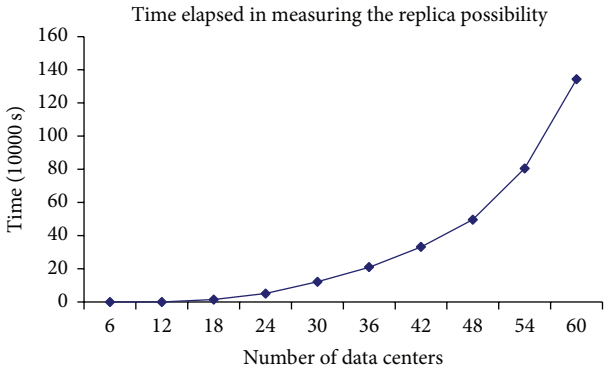


Figure 10: Elapsed time in measuring the replica possibility.

6.4. System Performance Comparisons of Replicas and No Replicas. The random simulations are conducted on randomly generated data sets of different sizes, generation times, and usage frequencies. In the simulations, we use a number of 50 data sets, each with a random size from 100 GB to 1 TB. And the usage frequency is also random from 1 to 10 times.

Simulation 1 (time complexity with one replica strategy). In the previous section, we have simulated the possibility of creating replicas in cloud platform. However, the algorithm used is exhaustive method, whose time complexity is very high. Figure 10 describes the elapsed time with increased number of nodes. From Figure 10, we can see that the time has a sharp increase. It is reasonable that we have to consider several data centers as the candidate place for replica.

Simulation 2 (comparisons of replicas numbers between MCRP and optimal strategy). Next, we evaluate the efficiency of MCRP strategy with the optimal solution in different number of usage frequencies. In the simulation, we assume that there is different number of data centers in cloud environment. And also data set usage frequency via each data center is random.

Figure 11 describes comparisons of the replicas numbers with different usage frequency, where we can see that the numbers beyond the optimal values are in tolerable limitations. Also, it is obvious that, with the increase of usage frequency, the amount of replica numbers increased quickly. The reason is that too much of data access leads to a big burden for network transmission in current conditions, such as network latency and bandwidth.

Simulation 3 (comparison of total cost between MCRP, optimal strategy, "each data center with one replica (ARS)," and "only original data set with no replica (NRS)" strategies). MCRP is a minimum-cost replicas placements strategy that can meet the system requirements under the condition of minimum overall data set cost. In this simulation, we will put the emphasis on the reduced cost between MCRP.

Figure 12 describes comparisons of total cost among different replicas placements strategies, where we can see that the total cost of MCRP is greatly decreased compared to other two straightforward strategies NRS and ARS. However, the total cost of MCRP is higher than the optimal strategy though the time complexity is lower.

From the above experimental and simulation results, the following conclusions can be drawn: (1) the proposed data sets replicas placements strategy effectively reduces the cost of application data set; (2) the proposed data replica strategy reduces the number of replicas; also (3) the proposed data replica strategy can effectively achieve system load balance by placing the popular data files according to the cost and user access history.

7. Conclusions and Future Works

In this paper we have investigated a model that simultaneously optimizes replicas placements from cost-effective view in the cloud. This model has a number of important applications in replication technology. Also, our current research, including experiments and simulations, is based on Amazon cloud's cost model, which can be reused by replacing the corresponding cost ratio. The data sets replicas placements strategy proposed in this paper is generic and dynamic, which can be used in any data intensive applications with different price models of cloud service. As presented in Section 5 and demonstrated in Section 6, minimum-cost replica placements strategy is close to the optimal cost benchmark, though it may not achieve the minimum cost. Therefore, it has great significance in theory and application explained as follows:

(1) The strategy proposed in this paper mainly focused on only one data set d_m, not all the data sets in a dynamic cloud computing system, which is simple and clear in practice. In this way, the popular data sets with high user frequencies are of particular concern, for the reason that they are important factors determining the system performance. However, for those data sets with low access frequency, even never used since generated, we have no need to consider their replicas.

(2) Considering the dynamic nature, which is the main metric of the cloud computing system, such as cost of transfer and storage and the user access frequency, the minimum-cost replica strategy is still available and

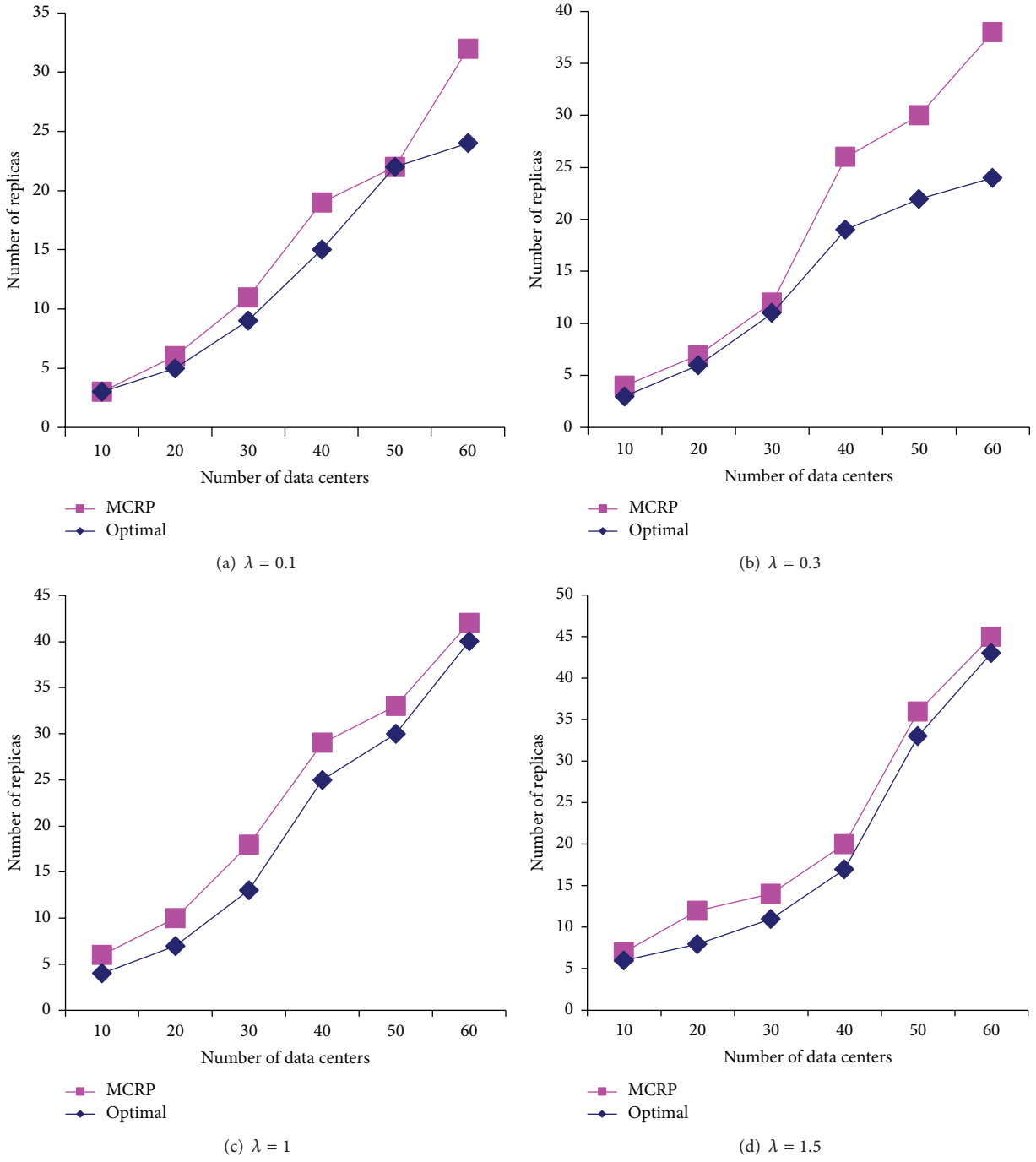


Figure 11: Comparisons of replicas numbers with different usage frequency.

effective, since we focus on a period of time T, not a time point t. In this way, the strategy is still applicable as long as we know the total cost in the past time.

(3) The replica deletion and maintenance strategies are also easy to obtain from the minimum-cost replica strategy. The basic idea is that when comparing the total cost with replica and the cost of no replica, then delete replicas once the cost with replicas is greater than no replica. On the other hand, we can update the replicas stored places according to the minimum-replicas strategy at scheduled time intervals.

Furthermore, experimental results and analysis show that the proposed strategy in cloud environment is feasible, effective, and universal. Hence, we deem that it is highly practical as a replica strategy. However, this paper presents the first attempt to apply the technique to solve the problem as how to place data sets replicas in the most appropriate data centers in the cloud from the minimum-cost view. It must

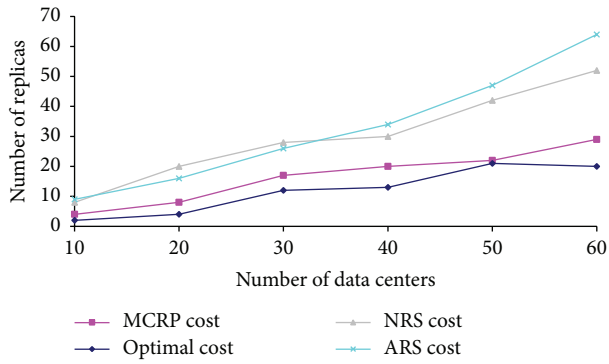


Figure 12: Comparisons of total cost among different strategies.

be kept in mind that these findings are the results of a preliminary study. To be more useful in practice, future works can be conducted from the following aspects:

(1) The current work in this paper has an assumption that the data set's usage frequencies are obtained from the system log files. Models of forecasting data set usage frequency can be further studied, with which our benchmarking approaches and replicas strategies can be adapted more widely to different types of applications.

(2) The replicas placements strategy should incorporate the data set generation, and deduplication technology, especially data content based deduplication technology, which is a strong and growing demand for business to be able to more cost-effectively manage big data while using cloud computing platforms.

Competing Interests

The author declares having no competing interests.

Acknowledgments

Some experiments in this paper were done on the cloud platform of SwinDeW-C in Swinburne University of Technology during Xiuguo WU's visiting period, which is sponsored by Shandong Provincial Education Department, China. Moreover, we want to show thanks to Doctor Dong Yuan and Professor Yun Yang, Swinburne University of Technology, Australia, for their valuable feedback on earlier drafts of this paper. In addition, this work presented in this paper is partly supported by Project of Shandong Province Higher Educational Science and Technology Program (no. J12LN33), China; the Doctor Foundation of Shandong University of Finance and Economics under Grant no. 2010034; and the Project of Jinan High-Tech Independent and Innovation (no. 201303015), China.

References

[1] Amazon S3, http://aws.amazon.com/s3.

[2] Google cloud storage, https://cloud.google.com/storage/.

[3] Windows Azure, https://www.azure.cn/.

[4] C.-H. Zuo, Z.-D. Lu, and R.-X. Li, "Load balancing in peer-to-peer systems using dynamic replication policy," *Journal of Chinese Computer Systems*, vol. 28, no. 11, pp. 2020–2024, 2007.

[5] T. Amjad, M. Sher, and A. Daud, "A survey of dynamic replication strategies for improving data availability in data grids," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 337–349, 2012.

[6] A. Martínez, F. J. Alfaro, J. L. Sánchez, F. J. Quiles, and J. Duato, "A new cost-effective technique for QoS support in clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 12, pp. 1714–1726, 2007.

[7] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proceedings of the Grid Computing Environments Workshop (GCE '08)*, pp. 1–10, IEEE, Austin, Tex, USA, November 2008.

[8] Y. Yang, K. Liu, J. Chen et al., "Peer-to-peer based grid workflow runtime environment of SwinDeW-G," in *Proceedings of the IEEE International Conference on e-Science and Grid Computing*, pp. 51–58, Bangalore, India, December 2007.

[9] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[10] S. Bansal, S. Sharma, I. Trivedi et al., "Improved self fused check pointing replication for handling multiple faults in cloud computing," *International Journal on Computer Science & Engineering*, vol. 4, no. 6, pp. 1146–1152, 2012.

[11] I. Arrieta-Salinas, J. E. Armendáriz-Iñigo, and J. Navarro, "Classic replication techniques on the cloud," in *Proceedings of the 7th International Conference on Availability, Reliability and Security (ARES '12)*, pp. 268–273, IEEE, Prague, Czech Republic, August 2012.

[12] W. Lloyd, M. J. Freedman, M. Kaminsky et al., "Don't settle for eventual: scalable causal consistency for wide-area storage with COPS," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, pp. 401–416, ACM, Cascais, Portugal, October 2011.

[13] W. Hoschek, J. Jaen-Martinez, A. Samar et al., "Data management in an international data grid project," in *Grid Computing—GRID 2000: First IEEE/ACM International Workshop Bangalore, India, December 17, 2000 Proceedings*, vol. 1971 of *Lecture Notes in Computer Science*, pp. 77–90, Springer, Berlin, Germany, 2000.

[14] D.-W. Chen, S.-T. Zhou, X.-Y. Ren, and Q. Kong, "Method for replica creation in data grids based on complex networks," *The Journal of China Universities of Posts and Telecommunications*, vol. 17, no. 4, pp. 110–115, 2010.

[15] X.-Y. Ren, R.-C. Wang, and Q. Kong, "Using optorsim to efficiently simulate replica placement strategies," *The Journal of China Universities of Posts and Telecommunications*, vol. 17, no. 1, pp. 111–119, 2010.

[16] M. Tu, P. Li, I.-L. Yen, B. Thuraisingham, and L. Khan, "Secure data objects replication in data grid," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 1, pp. 50–64, 2010.

[17] K. Ranganathan and I. Foster, "Identifying dynamic replication strategies for a high-performance data grid," in *Grid Computing—GRID 2001*, C. A. Lee, Ed., vol. 2242 of *Lecture Notes in Computer Science*, pp. 75–86, Springer, Berlin, Germany, 2001.

[18] R.-S. Chang, J.-S. Chang, and S.-Y. Lin, "Job scheduling and data replication on data grids," *Future Generation Computer Systems*, vol. 23, no. 7, pp. 846–860, 2007.

[19] K. Sashi and A. S. Thanamani, "Dynamic replication in a data grid using a modified BHR region based algorithm," *Future Generation Computer Systems*, vol. 27, no. 2, pp. 202–210, 2011.

[20] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving data availability through dynamic model-driven replication in large peer-to-peer communities," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, p. 376, IEEE, May 2002.

[21] R. M. Rahman, K. Barker, and R. Alhajj, "A predictive technique for replica selection in grid environment," in *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '07)*, pp. 163–170, Rio de Janeiro, Brazil, May 2007.

[22] V. Andronikou, K. Mamouras, K. Tserpes, D. Kyriazis, and T. Varvarigou, "Dynamic QoS-aware data replication in grid environments based on data 'importance'," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 544–553, 2012.

[23] K. Kalpakis, K. Dasgupta, and O. Wolfson, "Steiner-optimal data replication in tree networks with storage costs," in *Proceedings of the International Symposium on Database Engineering and Applications*, pp. 285–293, IEEE Computer Society, Grenoble, France, July 2001.

[24] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, K. Stockinger, and F. Zini, "Evaluation of an economy-based file replication strategy for a data grid," in *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '03)*, pp. 661–668, IEEE, May 2003.

[25] S. Melkote and M. S. Daskin, "An integrated model of facility location and transportation network design," *Transportation Research Part A: Policy and Practice*, vol. 35, no. 6, pp. 515–538, 2001.

[26] D. B. Shmoys, É. Tardos, and K. Aardal, "Approximation algorithms for facility location problems," in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97)*, pp. 265–274, ACM, May 1997.

[27] S. Melkote and M. S. Daskin, "Capacitated facility location/network design problems," *European Journal of Operational Research*, vol. 129, no. 3, pp. 481–495, 2001.

[28] Y. Yang, K. Liu, J. Chen, X. Liu, D. Yuan, and H. Jin, "An algorithm in SwinDeW-C for scheduling transaction-intensive cost-constrained cloud workflows," in *Proceedings of the IEEE Fourth International Conference on eScience (eScience '08)*, pp. 374–375, IEEE, Indianapolis, Ind, USA, December 2008.

[29] J. Yan, Y. Yang, and G. K. Raikundalia, "SwinDeW-a p2p-based decentralized workflow management system," *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 36, no. 5, pp. 922–935, 2006.

[30] Y. Yang, K. Liu, J. Chen, J. Lignier, and H. Jin, "Peer-to-peer based grid workflow runtime environment of SwinDeW-G," in *Proceedings of the IEEE International Conference on e-Science and Grid Computin*, pp. 51–58, Bangalore, India, December 2007.

[31] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A local-optimisation based strategy for cost-effective datasets storage of scientific applications in the cloud," in *Proceedings of the IEEE 4th International Conference on Cloud Computing (CLOUD '11)*, pp. 179–186, Washington, DC, USA, July 2011.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 9136107, 13 pages
http://dx.doi.org/10.1155/2016/9136107

Research Article

A Randomization Approach for Stochastic Workflow Scheduling in Clouds

Wei Zheng, Chen Wang, and Dongzhan Zhang

Department of Computer Science, School of Information Science and Engineering, Xiamen University, Xiamen 361005, China

Correspondence should be addressed to Dongzhan Zhang; zdz@xmu.edu.cn

Received 21 January 2016; Accepted 24 April 2016

Academic Editor: Laurence T. Yang

In cloud systems consisting of heterogeneous distributed resources, scheduling plays a key role to obtain good performance when complex applications are run. However, there is unavoidable error in predicting individual task execution times and data transmission times. When this error is being not negligible, deterministic scheduling approaches (i.e., scheduling based on accurate time prediction) may suffer. In this paper, we assume the error in time predictions is modelled in stochastic manner, and a novel randomization approach making use of the properties of random variables is proposed to improve deterministic scheduling. The randomization approach is applied to a classic deterministic scheduling heuristic, but its applicability is not limited to this one heuristic. Evaluation results obtained from extensive simulation show that the randomized scheduling approach can significantly outperform its static counterpart and the extra overhead introduced is not only controllable but also acceptable.

1. Introduction

Workflows have been widely used in various domains to depict complex computational application with multiple indivisible tasks and the data dependencies between tasks [1]. These workflows are normally derived from scientific problems in the fields of mathematics, astronomy, and so forth, for example, Montage [2]. As the massive requirements of calculation and communication became overwhelming in these disciplines, clusters and grids, as evolutionary forms of distributed computing, have been used to run workflow applications since the end of the 20th century [3, 4]. Recently, with more flexible and scalable capacity on computation and storage and less hardware/software installation expense, it has been gaining increasing popularity of using cloud computing infrastructure to run workflows [5–9].

The assignment of workflow tasks to computing resources, which is called *workflow scheduling*, is one of the key effects on the execution performance of the workflow in distributed computing environments like clouds. In general form of scheduling problems, a workflow is frequently represented by a Directed Acyclic Graph (DAG), where the nodes symbolize the tasks and the arcs with direction symbolize the data dependencies between tasks. The two terms "node" and "arc" will be interchangeably used in the rest of this paper. The most commonly focused objective of DAG scheduling is the minimization of the makespan (namely, overall execution time) of the workflow. Generally a DAG scheduling problem has been proven to be NP-Hard [10]. For getting closer favourable solution to this problem with acceptable time and space complexity, many researches have been carried out, and many heuristics have been proposed and published in the literature [11, 12].

Nevertheless, for majority of the existing DAG scheduling heuristics, the targeted DAG is modelled deterministically [13]. This means that the heuristics assumptions to the problem of inputs like task execution times and inter-task communication times are deterministic and precisely defined before. Clearly, the real-world workflow execution is much more complex than the above assumption because it is not possible to obtain accurate forecast of calculation and communication. Intuitively, the modelling of task computation times and communication times as random variables might be more realistic; it is also reasonabe to assume that the expected random variables and the variance can be predicted. In contrast to their deterministic counterparts,

the DAG scheduling problems modelled in stochastic fashion are called *stochastic scheduling* [14]. A big number of heuristics have been proposed for deterministic scheduling, but a small number for stochastic. There have been a plethora of studies showing deterministic DAG scheduling heuristics can hardly work well for their stochastic counterpart problems. This motivates the research on somehow adapting the existing deterministic DAG scheduling heuristics into the stochastic context and making improvement in terms of minimizing makespan, which is also the main focus of this paper.

In this paper, we consider the problem of scheduling a workflow onto a set of heterogeneous resources based on stochastic modelling of task execution times and task communication times with the objective of minimizing the makespan. For such a problem, a novel randomization scheduling approach is proposed. The proposed approach is applied to a classic deterministic scheduling heuristic and evaluated via extensive simulation experiments and the results exhibit a significant improvement of workflow execution performance with an acceptable extra overhead.

The rest of the paper is structured as follows. Related work is discussed in Section 2. The stochastic DAG scheduling problem and relevant definitions are presented in Section 3. The proposed randomization scheduling approach with an illustrative example is described in Section 4. The evaluation results are provided and discussed in Section 5. Finally, a conclusion is provided in Section 6.

2. Related Work

The past few decades witnessed a large number of efforts on developing various deterministic DAG scheduling heuristics, including HEFT [11], HBMCT [16], CPOP [11], GDL (DLS) [17], WBA [18], ILS [19], GA [20], SA [21], Triplet [22], TDS [23], STDS [24], and LDBS [25]. For an extensive list and classification of these heuristics we refer to [15, 26]. These heuristics differentiate with our work at the fact that they are based on deterministic scheduling model other than our stochastic model. Apparently, even when task execution times and data transmission times are modelled by some sort of random distribution, one can still apply one of the aforementioned deterministic scheduling heuristics by using the means of the random variables as inputs. However, as will be demonstrated later in this paper, this idea does not lead to a preferable schedule in most cases. Therefore, we derive a randomization approach by taking advantage of properties of random task execution times and data transmission times (as opposed to constant values). Although our approach can work with any deterministic heuristic in stochastic scheduling problems, we choose using the commonly cited and well-known deterministic heuristic: HEFT [11].

HEFT [11] is a deterministic list scheduling heuristic which aims to minimize the makespan of DAG applications on a bounded number of heterogeneous resources. The heuristic consists of two phases. In the first phase, the heuristic computes numeric ranks for all tasks based on their execution time predictions and data dependencies and then prioritizes tasks in a list. In the second phase, by the prioritized order, each task is allocated in turn to the resource which is estimated to minimize finished time of the task. It is worth mentioning that HEFT allows a task to be inserted into the existing task queue of a resource as long as the task dependency is not violated.

It has been widely recognized that due to the inaccuracy in time prediction deterministic scheduling heuristic relying on constant input may result in bad decision [27]. Over the recent years, some works have been carried out to evaluate the performance of deterministic DAG scheduling heuristics with stochastic model, such as [26, 28]. However, the main focus of these works is not on proposing an efficient scheduling heuristic for the stochastic DAG scheduling problem. One of recently popular ideas of addressing stochastic DAG scheduling problems is adapting the existing deterministic heuristics by changing their ranking function and/or the way of comparing task attributes. Examples can be found in [29–31]. Nevertheless, these heuristics still make scheduling decision in a deterministic manner. In contrast, our heuristic is randomized.

In our previous study [15], a Monte-Carlo based approach has been applied to the stochastic DAG scheduling problem to acquire a schedule which can outperform schedules generated by other comparable means. In essence, this approach relies on a significant amount of random searches on the solution space as well as an extensive evaluation for picking up the result schedule. As a result, a result schedule with a reasonable performance requires a considerable overhead. This paper is an extension to our previous work [32]. In this paper, we only do local search around a well-crafted schedule and the cost of evaluation of the searching results is trivial. That is to say, the overhead of the heuristic proposed in this paper is much less than the Monte-Carlo based approach, while a significant improvement on makespan can still be achieved.

3. Problem Description

3.1. Application Model. In this paper, a workflow application is represented by a Directed Acyclic Graph (DAG) $G = \{V, E\}$, where V denotes a set of interdependent tasks, each of which is represented by v_i, that is, $V = \{v_i \mid i = 1, 2, \ldots, n\}$, and E denotes a set of directed arcs, each of which represents data dependency between two tasks, that is, $E = \{e_{j,k}\}$. For instance, $e_{j,k} = v_j \rightarrow v_k$ means the input of task v_k depends on the output of task v_j. In this case, v_k is a child of v_j and v_j is a parent of v_k; moreover, v_k cannot start to run before receiving all necessary data from its parents. The node without parents is called entry node, and the node without children exit node. For the sake of standardization we assume that every DAG here has only one single entry node and one single exit node. For illustration, Figure 1 presents an example of DAG with 10 nodes.

3.2. Platform Model. We assume that the underlying computing platform comprises a fixed number of heterogeneous resources and uses $R = \{r_1, r_2, \ldots, r_m\}$ to denote the set of resources. A task v can be executed at any resource p; however a resource cannot execute more than one task at a time. In addition, no temporal interruption is allowed

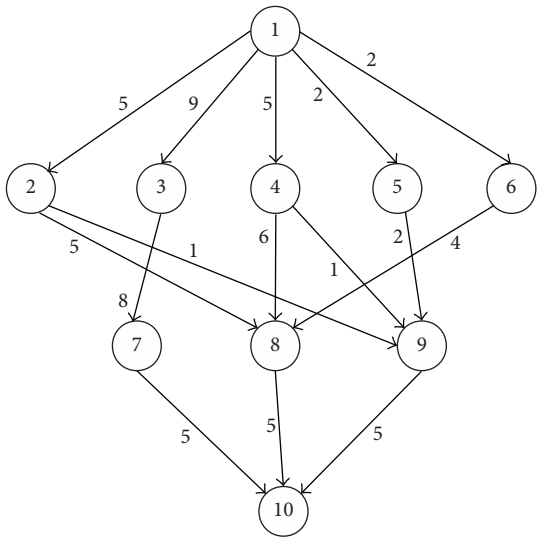


Figure 1: A DAG example.

during this execution. There is a dedicated dual-direction communication link between every pair of resources. This means that the tasks allocated onto different resources can transmit data to each other with no contention. For any random variable RV, we use $\mu(\mathrm{RV})$ and $\sigma(\mathrm{RV})$ to represent the expectation and the standard deviation of RV. Given that $\mathrm{ET}_{i,p}$ represents the random variable modelling the execution time of task v_i on resource r_p and $\mathrm{CT}_{i,j,p,q}$ the time for transmitting data from v_i located on r_p to v_j located on r_q, we assume that for each i and p ($1 \le i \le n$, $1 \le p \le m$) $\mu(\mathrm{ET}_{i,p})$ and $\sigma(\mathrm{ET}_{i,p})$ are known. We also assume the data amount transmitted on each edge (denoted by $\delta_{i,j}$), and the average time needed for transmitting one unit of data from one resource to another (denoted by $\gamma_{p,q}$) is known, so

$$\mu\left(\mathrm{CT}_{i,j,p,q}\right) = \delta_{i,j} \cdot \gamma_{p,q}, \tag{1}$$

as well as $\sigma(\mathrm{CT}_{i,j,p,q})$, is known. For illustration, Tables 1(a) and 1(b) show the stochastic model of task execution times and communication times of the DAG depicted in Figure 1. Table 2 shows an example of possible outcome of the stochastic model presented in Tables 1(a) and 1(b).

We use $\mathrm{ST}_{j,q}$ to denote the start time of task v_j on resource r_q and $\mathrm{FT}_{j,q}$ the finish time. In addition, we let $\mathrm{pre}(j)$ and $\mathrm{suc}(j)$ stand for the set of parents and children of v_j, respectively, and $\mathrm{alloc}(j)$ symbolize the resource where v_j is allocated. Obviously, we have

$$\mathrm{FT}_{j,q} = \mathrm{ST}_{j,q} + \mathrm{ET}_{j,q}. \tag{2}$$

In addition,

$$\mathrm{ST}_{j,q} = \max\left\{\mathrm{FT}_{i,p} + \mathrm{CT}_{i,j,p,q} \mid i \in \mathrm{pre}\left(j\right),\ p = \mathrm{alloc}\left(i\right)\right\}. \tag{3}$$

Table 1: Stochastic model of the DAG example shown in Figure 1.

(a) Stochastic model of execution times

	Resource 1		Resource 2		Resource 3	
	Exp. val.	Std. dev.	Exp. val.	Std. dev.	Exp. val.	Std. dev.
Task 1	24.00	4.00	2.00	0.33	7.00	1.17
Task 2	36.00	6.00	32.00	5.33	49.00	8.17
Task 3	42.00	7.00	49.00	8.17	12.00	2.00
Task 4	12.00	2.00	8.00	1.33	56.00	9.33
Task 5	81.00	13.50	20.00	3.33	16.00	2.67
Task 6	9.00	1.50	30.00	5.00	24.00	4.00
Task 7	48.00	8.00	4.00	0.67	8.00	1.33
Task 8	64.00	10.67	18.00	3.00	42.00	7.00
Task 9	36.00	6.00	16.00	2.67	1.00	0.17
Task 10	54.00	9.00	28.00	4.67	63.00	10.50

(b) Stochastic model of communication times

	Resource 1		Resource 2		Resource 3	
	Exp. val.	Std. dev.	Exp. val.	Std. dev.	Exp. val.	Std. dev.
Resource 1	0.00	0	2.00	0.33	8.00	1.33
Resource 2	2.00	0.33	0.00	0.00	4.00	0.67
Resource 3	8.00	1.33	4.00	0.67	0.00	0.00

Table 2: A sampling of the stochastic model shown in Table 1.

(a) A sampling of stochastic task execution times

	Resource 1	Resource 2	Resource 3
Task 1	13.45	2.05	6.75
Task 2	33.44	22.64	45.42
Task 3	25.05	51.33	11.61
Task 4	14.82	8.85	47.48
Task 5	88.77	22.64	15.39
Task 6	6.59	25.23	26.17
Task 7	35.47	4.05	9.08
Task 8	54.31	22.29	50.60
Task 9	33.52	19.80	1.28
Task 10	56.57	28.59	77.36

(b) A sampling of stochastic task communication times

	Resource 1	Resource 2	Resource 3
Resource 1	0.00	2.58	8.46
Resource 2	2.58	0.00	5.58
Resource 3	8.46	5.58	0.00

3.3. Problem Definition. The main objective of this paper is to minimize the overall execution time of a DAG application. Given that the start time of entry node is always time 0, based on the definitions and assumptions presented above, the problem to be addressed in this paper is to generate a schedule S, which specifies the mapping of tasks and resources, as well as the execution order of tasks on each resource, so that the random variable $\mathrm{FT}_{\mathrm{exit\_node},\,\mathrm{alloc(exit\_node)}}$ can be minimized.

Input: DAG G on a set of resources R with stochastic model
Output: A schedule specifying task mapping and execution order
(1) Create a candidate schedule list $\mathscr{L}$, which is initially empty.
(2) Run a deterministic heuristic DH to generate the schedule entirely based on all mean values of task execution times and communication times, and put the schedule into $\mathscr{L}$.
(3) **while** the termination condition of the producing phase is not met **repeat**
(4) Run the randomized heuristic RH and push the result schedule into $\mathscr{L}$.
(5) **endwhile**
(6) Compute the expected makespan based on mean values of all stochastic inputs for each schedule in $\mathscr{L}$.
(7) **Return** the schedule with the minimum expected makespan as the result schedule.

Algorithm 1: The outline of the randomization scheduling approach.

4. Method

In this section, we firstly present the basic idea of the proposed randomization approach and then describe the details of the randomized HEFT heuristic (RHEFT), which is derived from the combination of our randomization approach and the classic HEFT heuristic. Furthermore, an enhanced version of RHEFT (named, ERHEFT) is proposed.

4.1. Basic Idea. Among different categories of deterministic DAG scheduling heuristics, list scheduling seems receiving most research attention, because this kind of heuristics can usually obtain a reasonably good schedule result without too much overhead. As mentioned in Section 1, deterministic DAG scheduling heuristics receive constant time prediction as inputs. List scheduling heuristics firstly sort all workflow tasks in terms of a rank, which is computed based on the time prediction and associated with each task, and then allocate the sorted tasks one after another onto a specific resource. The resource where a task is allocated is normally decided by which resource can minimize a certain time-related attribute (e.g., estimated finish time of the current task) that can be calculated based on the time prediction and is distinctive on different resources. By doing so, list scheduling aims at a good trade-off between optimization and algorithm complexity. However, there is no guarantee that the resource allocation made is the best possible decision for minimizing makespan.

The issue is more complicated in the case of stochastic scheduling model. It will be more doubtable whether the scheduling decision made by deterministic list scheduling results will be favourable for minimizing the makespan, because the time prediction based on which decision is made is unreliable.

Assuming the time prediction can be modelled by probability distribution, our hypothesis is that it is almost impossible to develop a static algorithm which can always obtain an optimal schedule. Therefore, we consider generating a set of various schedules based on the random prediction and pick up that one which has the best chance to win. As the time prediction is modelled randomly, when comparing two time-related variables, for example, task finish times of different tasks, there is no certain result. To decide which task finished earlier, we need to roll a dice. Apparently, by randomly deciding the comparison result of task finish times, which is important factors for making scheduling decision, we will generate various scheduling results. We regard all these schedules as candidates and the one which has the smallest expectation of makespan as the final output of our approach.

The outline of our randomization scheduling approach is presented in Algorithm 1, where DH denotes a given deterministic DAG scheduling heuristic and RH means a randomized scheduling heuristic.

4.2. The Randomized HEFT Heuristic (RHEFT). Among existing list scheduling heuristics, the most well-known and commonly cited one is heterogeneous-earliest-finish-time (HEFT) heuristic [11]. We thereby choose to apply the aforementioned randomization approach to HEFT and propose a novel approach named RHEFT which is based on a randomized HEFT. Namely, we let DH be HEFT and RH the randomized HEFT.

The details of the randomized HEFT heuristic are presented in Algorithm 2. Similar to HEFT, the randomized HEFT has two phases. In the first phase (Algorithm 2, lines: 1-2), the upward ranking of each node (denoted by Urank) is computed as follows:

$$\mathrm{Urank}\left(v_i\right) = \overline{\mathrm{ET}_i} + \max_{v_j \in \mathrm{suc}(i)} \left\{\overline{\mathrm{CT}_{i,j}} + \mathrm{Urank}\left(v_j\right)\right\}, \tag{4}$$

where

$$\overline{\mathrm{ET}_i} = \frac{\sum_{p=1}^{m} \mu\left(\mathrm{ET}_{i,p}\right)}{m},$$
$$\overline{\mathrm{CT}_{i,j}} = \frac{\sum_{p=1}^{m} \sum_{q=1}^{m} \mu\left(\mathrm{CT}_{i,j,p,q}\right)}{m \cdot m}. \tag{5}$$

Especially for exit node, we have $\mathrm{Urank}(v_{\mathrm{exit\_node}}) = \overline{\mathrm{ET}_{\mathrm{exit\_node}}}$.

In the second phase of the randomized HEFT (Algorithm 2, lines: 3–9), it is needed to compute for each node on each resource the earliest estimate start time (denoted by

(1) Compute Urank (as defined in (4)) for all tasks.
(2) Sort all tasks in a list $\mathscr{L}$ in the non-descending order of Urank.
(3) **for** $k \coloneqq 1$ to n **do** (where n is the number of tasks)
(4) Select the kth task v^* from the list $\mathscr{L}$.
(5) **for** each resource $r_p \in R$ **do**
(6) Compute the mean value and variance of estimated finish time of v^* on r_p (as defined in (7)).
(7) **endfor**
(8) Decide the winner resource of estimated finish time (r^*) for v^* according to the *random comparison policy*.
(9) Allocate v^* to r^*.
(10) **endfor**

Algorithm 2: The randomized HEFT heuristic.

EST) and the earliest estimate finish time (denoted by EFT), which are defined as follows:

$$\mathrm{EST}_{j,q} = \max\left\{\mathrm{AVT}_q, \max_{v_i \in \mathrm{pre}(j)}\left\{\mathrm{EFT}_{i,p} + \mathrm{CT}_{i,j,p,q}\right\}\right\}, \quad (6)$$

$$\mathrm{EFT}_{j,q} = \mathrm{EST}_{j,q} + \mathrm{ET}_{j,q}, \quad (7)$$

where AVT_q is the earliest available time of resource r_q to allocate v_j, taking into account the current load of r_q and the estimate execution time of v_j on r_q (i.e., $\omega(\mathrm{ET}_{j,q})$). In addition, $r_p = \mathrm{alloc}(i)$. Apparently, at this moment, v_i has already been allocated. Particularly, for the entry node, $\mathrm{EST}_{\mathrm{entry\_node},q} = 0$ for every r_q. The main difference between the randomized HEFT and its initial version is the random comparison policy mentioned in line 8. For each task, we use this comparison policy to compare the estimated finish times (i.e., EFT) of different resources, which are random variables in our stochastic model, as the estimated finish time is determined by summing up the task execution times and the communication times along the critical path. Let CP denote the critical path, ND the set of nodes, and ED the set of edges along with CP; then we have the earliest finish time as below:

$$\mathrm{EFT} = \sum_{i \in \mathrm{ND}} \mathrm{ET}_i + \sum_{j \in \mathrm{ED}} \mathrm{CT}_j. \quad (8)$$

For ease of analysis, we assume all task execution times and communication times follow normal distribution. Then we have

$$\begin{aligned} \mu(\mathrm{EFT}) &= \sum_{i \in \mathrm{ND}} \mu\left(\mathrm{ET}_i\right) + \sum_{j \in \mathrm{ED}} \mu\left(\mathrm{CT}_j\right), \\ \sigma(\mathrm{EFT})^2 &= \sum_{i \in \mathrm{ND}} \left(\sigma\left(\mathrm{ET}_i\right)^2\right) + \sum_{j \in \mathrm{ED}} \left(\sigma\left(\mathrm{CT}_i\right)^2\right). \end{aligned} \quad (9)$$

Say there are two random variables EFT and EFT′ to compare and determine which is larger. We let DV = EFT − EFT′. Then we have

$$\begin{aligned} \mu(\mathrm{DV}) &= \mu(\mathrm{EFT}) - \mu\left(\mathrm{EFT}'\right), \\ \sigma(\mathrm{DV})^2 &= \sigma(\mathrm{EFT})^2 + \sigma\left(\mathrm{EFT}'\right)^2. \end{aligned} \quad (10)$$

Apparently, the value of DV corresponds to the comparison result of EFT and EFT′. The key idea of our random comparison policy is to make a random draw of DV value, if the outcome is less than zero, EFT < EFT′; otherwise, EFT ≥ EFT′. However, to implement a precise random draw of DV is not easy. Therefore, we use μ(DV) and σ(DV) for approximation. In detail, without losing generality, we assume $\mu(\mathrm{DV}) > 0$, and if $2 \times \sigma(\mathrm{DV}) > \mu(\mathrm{DV})$,

$$P(\mathrm{DV} < 0) = \frac{2 \times \sigma(\mathrm{DV}) - \mu(\mathrm{DV})}{4 \times \sigma(\mathrm{DV})}; \quad (11)$$

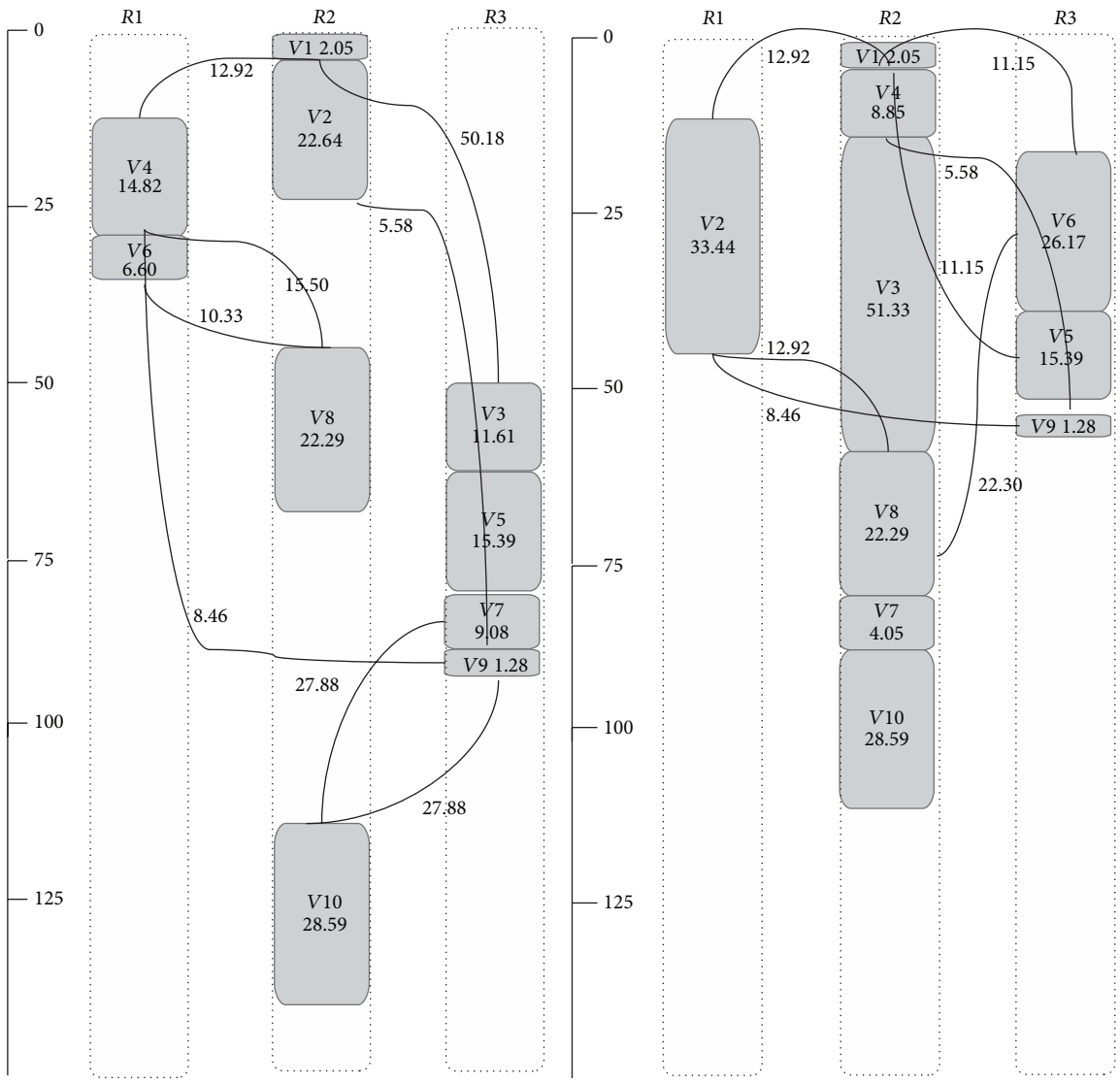
else $P(\mathrm{DV} < 0) = 0$. One can easily get $P(\mathrm{DV} \geq 0) = 1 - P(\mathrm{DV} < 0)$. That is to say, the comparison result between EFT and EFT′ is a 0-1 random variable. With probability of $P(\mathrm{DV} < 0)$, the outcome is EFT < EFT′; with probability of $P(\mathrm{DV} \geq 0)$, EFT ≥ EFT′. By this random comparison policy, the deterministic HEFT is randomized. Note that as described in Algorithm 1 (lines 3–5), RHEFT requires running the randomized HEFT for a certain number of times to generate candidate schedules.

For illustration purpose, we run HEFT and RHEFT to the DAG example modelled by Figure 1 and Table 1 and obtain scheduling results, respectively. For RHEFT, the scheduling result is picked up from 10 candidates. In order to present these scheduling results, we assume the values shown in Table 2 are the real task execution times and data transmission times. Then the scheduling details can be depicted by Figure 2. In this case, RHEFT obtains a makespan of 117.17, which makes a significant improvement (19.8%) to HEFT with a makespan of 146.15.

4.3. Further Investigation on RHEFT. Although the illustrative example shows that RHEFT can significantly outperform HEFT on minimizing makespan, there are still two open issues regarding with the configuration of RHEFT which are worthy of further investigation:

(i) In RHEFT, how many candidate schedules should we generate to gain substantial improvement on makespan without paying too much overhead?

(ii) How often and to what extent can RHEFT improve the makespan obtained by HEFT over various DAG examples?



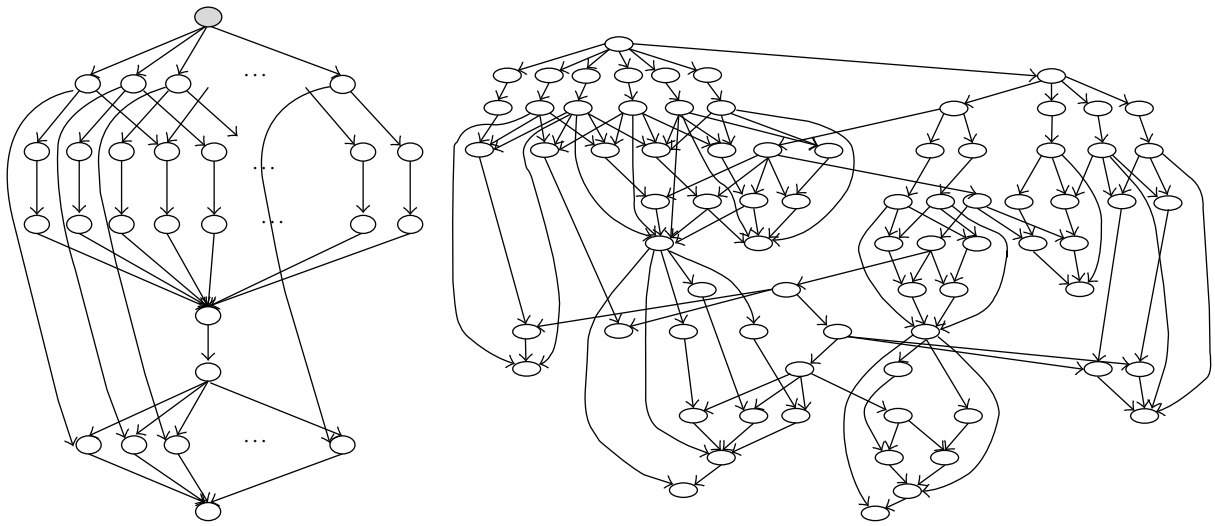
(a) HEFT (b) RHEFT

Figure 2: Scheduling result of the DAG example shown in Figure 1 by using (a) HEFT; (b) RHEFT.

We examine these two issues by evaluation in the rest of this subsection.

4.3.1. Evaluation Setting. We built a simulated computing system and a stochastic DAG generator, which allow various parameter settings. We use two types of DAGs: Montage [2] and LIGO [33] as shown in Figure 3, which are derived from real-world applications and have distinctive structures and sizes. We consider the number of resources used to be 3 and 8. All random variables used in our stochastic model are assumed to follow normal distribution. For the execution time of each task on each resource, we randomly select its expected value from the range of $[1, 100]$ and its standard deviation value as 1/6 of its expected value. Similar setting is applied to the communication time between tasks on different resources. We specify a parameter named communication-computation-ratio (CCR), which means the ratio between the average communication cost and the computation one, and adjust the value of communication times to meet the specified CCR value. We randomly specify the CCR value from $[0.5, 1.5]$ in our experiments.

4.3.2. How Many Candidate Schedules Are Needed? One can easily imagine that the more times RHEFT runs the randomized HEFT heuristic, the more likely a better candidate schedule may be obtained, and on the other hand the more time cost is needed to be paid. In order to examine how many times RHEFT should repeat running the randomized scheduling procedure, we specify four DAG instances and observe the expected makespan RHEFT can gain for these DAGs as the number of repetition grows. Figure 4 shows the observation results.



(a) Montage, 34 nodes (b) LIGO, 77 nodes

Figure 3: DAG applications used in the evaluation [15].

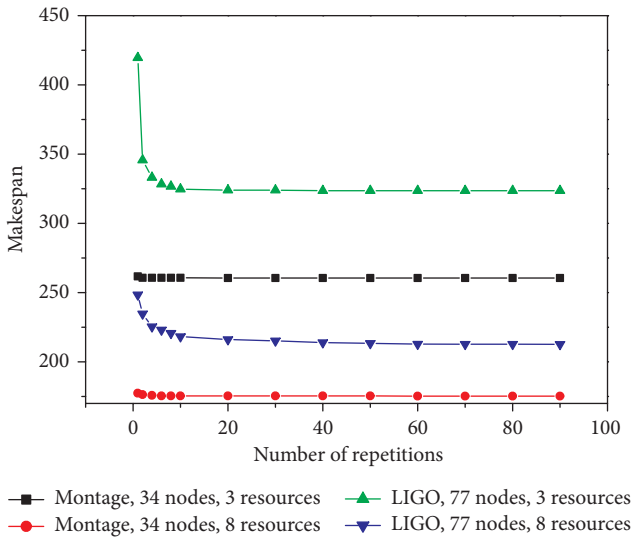


Figure 4: Change of expected makespan as the number of candidates generated by RHEFT increases.

In the diagram shown in Figure 4, RHEFT with zero repetition actually boils down to HEFT. As we can see from the curves denoting "Montage on 3 resources" and "LIGO on 8 resources," the expected makespan of RHEFT reduces rapidly as the number of repetitions grows. After the number of repetitions reaches 10, no significant improvement can be observed on the expected makespan. This observation indicates that RHEFT is promising, as it can generate a candidate schedule which is fairly better than HEFT's with only few more repetitions.

4.3.3. Improvement Rate on Makespan. Next, we observe to what extent can a RHEFT schedule improve a HEFT schedule on the expected makespan. Again, we consider the DAG type to be Montage with 34 nodes and LIGO with 77 nodes and the number of resources to be 3 and 8. Then for each combination of the DAG type and the number of resources, we generate 100 instances of the stochastic model. For each instance, we collect the expected makespans obtained by RHEFT and HEFT, respectively. For comparison, we define the metric "improvement rate," which is the ratio between the reduced expected makespan and the expected makespan of HEFT. Figure 5 shows the results of improvement rate. In general, RHEFT obtains significant improvement on the expected makespan (above 20%) in the case where LIGO is used. Nevertheless, when Montage is used, RHEFT obtains a similar result in the majority of cases of the 100 instances. It can also be seen that the advantage of RHEFT over HEFT may be weakened as the number of resources increases. Anyway, with every setting of DAG type and resource number, there is always a chance that RHEFT can reduce the expected makespan more than 20%.

4.4. The Enhanced RHEFT Heuristic. By making random decision in the resource allocation phase of HEFT, we derive a novel scheduling approach RHEFT which significantly outperforms HEFT on minimizing makespan. This encourages us to extend RHEFT by making random decision when prioritizing the tasks in the listing phase.

The extension is fairly straightforward. In the first phase of the randomized HEFT (Algorithm 2, lines: 1-2), instead of defining task rank by the constant value Urank (as defined in (4)), we consider it as a random variable Rrank. The definition of Rrank is somehow correlated with Urank.

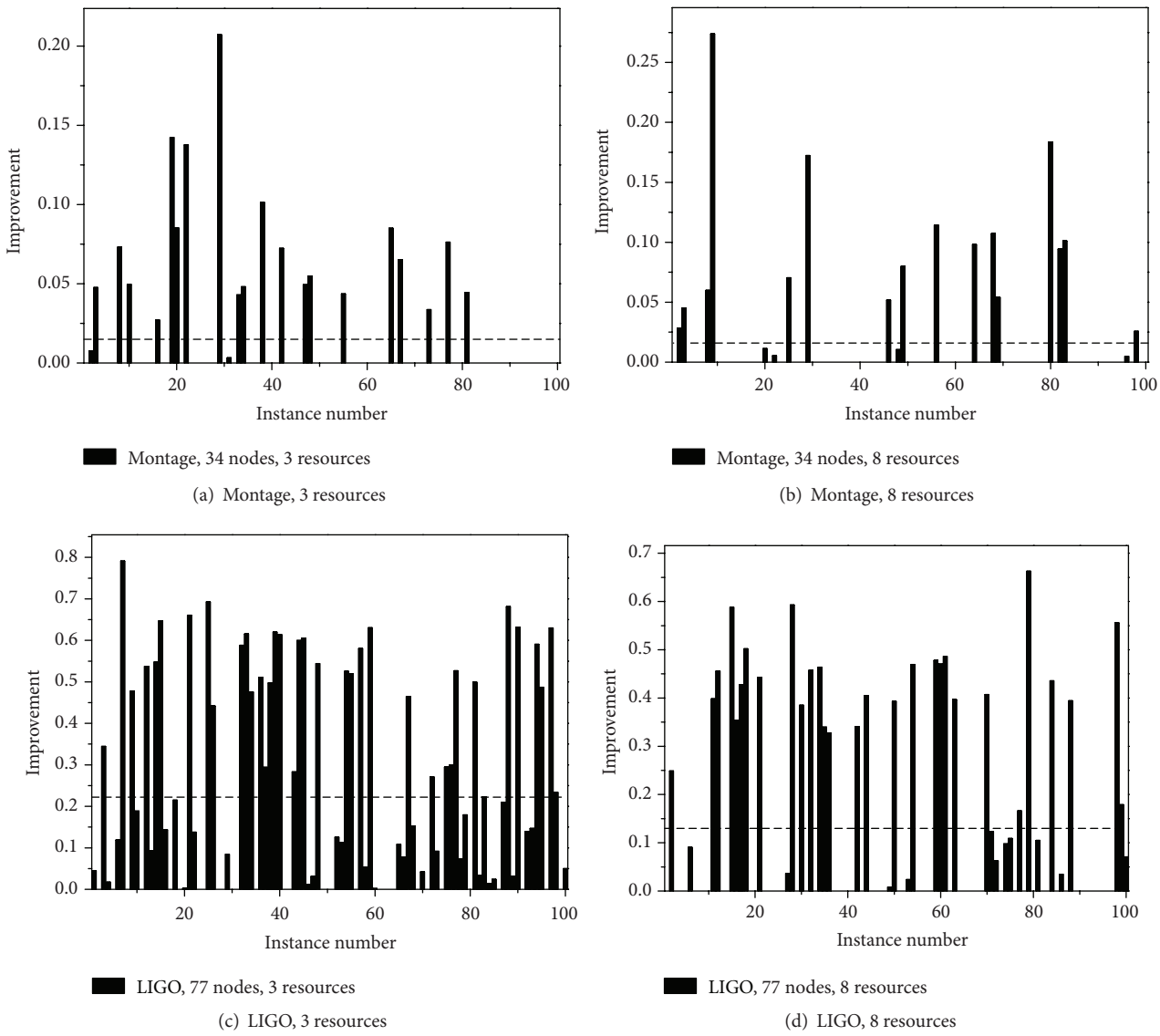


Figure 5: Makespan improvement of RHEFT over HEFT over 100 instances.

Recall that by (4), for task v_i, $\mathrm{Urank}(v_i)$ is calculated by accumulating the time costs associated with the nodes and edges along the critical path from v_i to the exit node. Let Ph_i denote this critical path for v_i, N_i the set of nodes, and E_i the set of edges along with Ph_i; then we have the definition of $\mathrm{Rrank}(v_i)$ as below:

$$\mathrm{Rrank}\left(v_i\right) = \sum_{i \in N_i} \mathrm{MET}_i + \sum_{j \in E_i} \mathrm{MCT}_j, \tag{12}$$

where

$$\mathrm{MET}_i = \frac{\sum_{p=1}^{m} \mathrm{ET}_{i,p}}{m}, \qquad \mathrm{MCT}_j = \frac{\sum_{p=1}^{m} \sum_{q=1}^{m} \mathrm{CT}_{j,p,q}}{m \cdot m}. \tag{13}$$

Especially for exit node, we have $\mathrm{Rrank}(v_{\mathrm{exit\_node}}) = \mathrm{MET}_{\mathrm{exit\_node}}$.

When prioritizing tasks in the listing phase, we need to compare the rank of v_i with v_j. The comparison procedure, which is named *randomized prioritizing procedure*, is carried out as follows:

(1) We firstly examine if there is any task dependency between v_i and v_j.

(2) If v_i is an ancestor of v_j, v_i should be given higher priority and placed before v_j in the list.

(3) If there is no dependency between v_i and v_j, random variables $\mathrm{Rrank}(v_i)$ and $\mathrm{Rrank}(v_j)$ will be compared by the random comparison policy as described in Section 4 to determine which one has the higher priority.

(1) Compute Rrank (as defined in (12)) for all tasks.
(2) Sort all tasks in a list $\mathscr{L}$ in the non-descending order of Rrank according to the *randomized prioritizing procedure*.
(3) **for** $k \coloneqq 1$ to n **do** (where n is the number of tasks)
(4) Select the kth task v^* from the list $\mathscr{L}$.
(5) **for** each resource $r_p \in R$ **do**
(6) Compute the mean value and variance of estimated finish time of v^* on r_p (as defined in (7)).
(7) **endfor**
(8) Decide the winner resource of estimated finish time (r^*) for v^* according to the *random comparison policy*.
(9) Allocate v^* to r^*.
(10) **endfor**

Algorithm 3: The enhanced randomized HEFT heuristic.

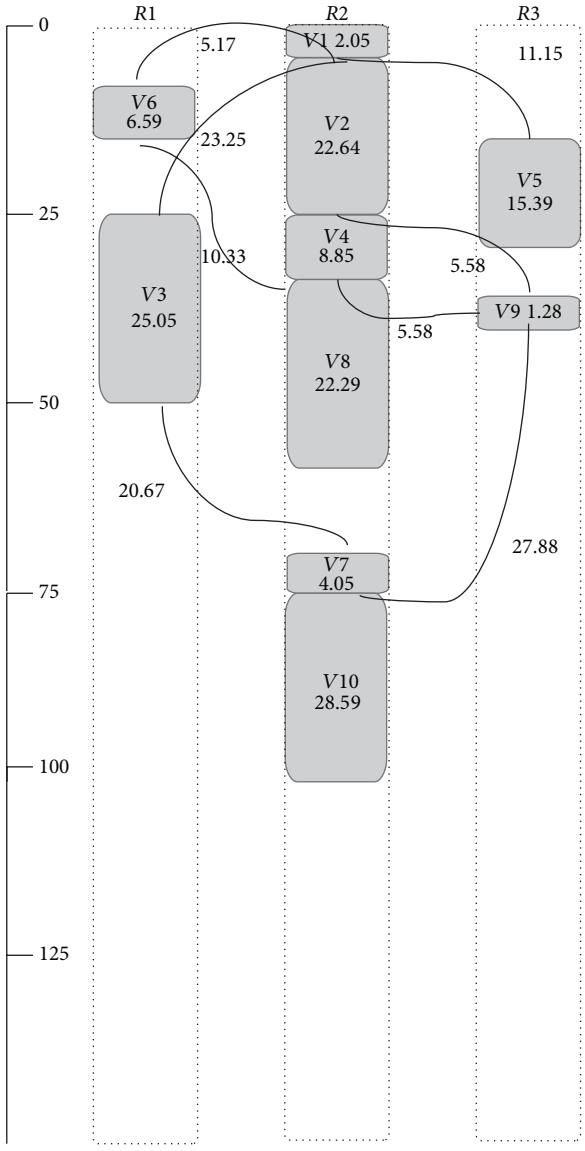


Figure 6: Scheduling result of the DAG example shown in Figure 1 by using ERHEFT.

We apply the above procedure to the listing phase of RHEFT and then derive the enhanced RHEFT heuristic, namely, ERHEFT. The details of the enhanced randomized HEFT heuristic are provided in Algorithm 3.

Apparently, when ERHEFT generates candidate schedules, the prioritized task list may be different from that of RHEFT. This makes it possible for ERHEFT to generate more candidate schedules than RHEFT can do. As a result, a schedule with better makespan may be obtained. For illustration, Figure 6 shows the scheduling result by applying ERHEFT to the DAG example modelled by Figure 1 and Table 1. This schedule has a makespan of 103.67 which is better than the schedule acquired by RHEFT as shown in Figure 2(b).

Similar to the way by which we investigate in Section 4.3.2, we observe how the expected makespan of ERHEFT changes as the number of repetitions used increases. Here, the same evaluation setting as specified in Section 4.3.2 is used and the result is shown in Figure 7. This result indicates that 200 may be the appropriate number of repetitions that should be used by ERHEFT.

5. Evaluation

In order to compare the performance of HEFT, RHEFT, and ERHEFT in stochastic scheduling model, we adopt the evaluation setting as mentioned in Section 4.3.1 and evaluate the expected makespan and the time cost of the competitors with different configurations of evaluation parameters. The machine we used to carry out the evaluation has the following hardware configuration: CPU Intel I3-4130 3.40 GHz, 4 G DDR3 memory, and 500 G hard disk. We used Java (JDK 1.7) to implement all heuristics and the simulation.

First, we evaluate the average makespan that HEFT, RHEFT, and ERHEFT can obtain with stochastic model. For each experiment, we firstly specify the DAG type and the number of resources we are going to use. Then we generate the expected value and variance for stochastically modelling

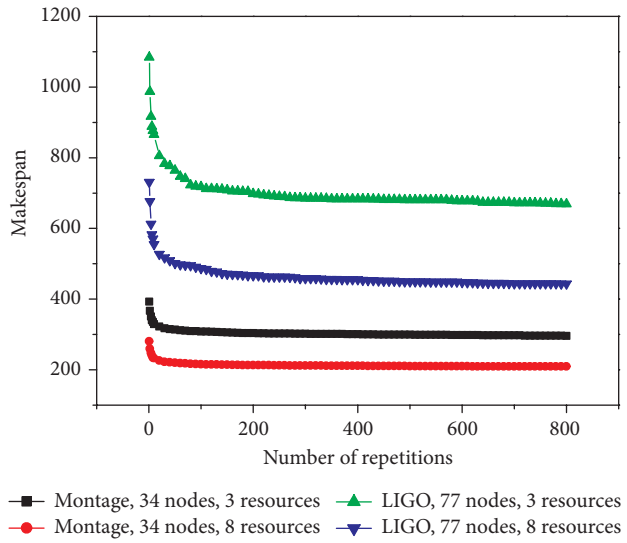


Figure 7: Change of expected makespan as the number of candidates generated by ERHEFT increases.

each task execution time and then generate communication times with specified CCR value, which as a whole is called a stochastic model of the given DAG and resources. For each combination of a DAG and a set of resources, we generate 100 stochastic models. And for each stochastic model, we run HEFT, RHEFT, and ERHEFT and obtain their schedules, respectively. Each time we take a sample for the stochastic model (as shown in Table 2, including all execution times and communication times), we can view them as runtime information gathered after the DAG application completes and use them to evaluate the performance of each schedule. To be fair, we take 100 samples from each stochastic model. So the result for each compared heuristic on a given DAG and given resources is averaged over 10000 experiments (100 stochastic models, each of which has 100 samples) in total. For comparison, we use the metric of "speedup" which is defined as the ratio between the average sequential execution time of all tasks and the makespan obtained by a heuristic.

In order to compare the heuristic competitors in different scenario, we consider the number of resources to be 3, 6, and 8 and collect the average speedup results for CCR being equal to 0.1, 1, and 10, respectively. The collected results are shown in Figure 8.

One can easily see that in all combination of evaluation parameter settings ERHEFT has better average speedup than RHEFT, while RHEFT has better average speedup than HEFT. The improvement on average speedup of ERHEFT over HEFT can be up to around 20% in the case where Montage is executed on 3 resources and CCR = 0.1 is used. It is interesting to see that the effectiveness of ERHEFT and RHEFT is closely related to CCR. When CCR is high, ERHEFT and RHEFT usually achieve more significant improvement on average speedup over HEFT. This indicates our randomization approach may work better with workflow applications which are data intensive. However, when CCR turns to be as low as 0.1, the difference in the average speedup obtained by HEFT, RHEFT, and ERHEFT seems trivial. From a different perspective, this may also imply that HEFT works especially well with computation intensive applications and thus leave little space for further improving its makespan.

In addition, we measure the time cost needed by HEFT, RHEFT, and ERHEFT with different sizes of Montage DAG and different numbers of resources. We use the ratio of the time cost of RHEFT (ERHEFT) over that of HEFT as the metric and the measurement result is shown in Figure 9. From the diagram we can see that in most of the cases the ratio of RHEFT over HEFT is within the range of 5 to 15. Moreover, there is no rapid ascending trend as the DAG size or the number of resources, which represents the scale of the scheduling problem, grows. This indicates that the time complexity of RHEFT is close to HEFT. Because HEFT usually needs very little time to compute a schedule, the additional overhead introduced by a certain number of loops of running HEFT and extra computation of random variables, which results in 5 to 15 times of the time cost of HEFT, is acceptable. The radio of ERHEFT over HEFT exhibits a curve similar to RHEFT over HEFT. Even though the radio of ERHEFT over HEFT reaches the range of 100 to 500, as the time cost for a single execution of HEFT is tiny, the overall time cost for ERHEFT is still acceptable. For instance, in our empirical results, for DAG with 234 nodes ERHEFT runs for 3.2 seconds while for 12 resources ERHEFT runs for only 1.9 seconds. Moreover, by adjusting the number of loops used in the RHEFT approach, we can flexibly get a good trade-off between the scheduling performance and the heuristic overhead.

6. Conclusion

In this paper, we explore into the problem of scheduling workflow tasks onto a set of heterogeneous cloud resources with stochastic model of task execution and communication. We attempt to extend deterministic DAG scheduling heuristic, to gain better average makespan. As a progress, a randomization scheduling approach is proposed. We apply the randomization approach to the classic deterministic heuristic HEFT and two versions of novel randomized heuristic: RHEFT and ERHEFT, are produced. We evaluate and compare the performance of HEFT, RHEFT, and ERHEFT with extensive simulation experiments where two real-world workflow applications are used. The experimental results suggest that RHEFT and ERHEFT are both promising for stochastic workflow scheduling, as RHEFT and ERHEFT not only significantly reduce the average makespan in most cases of experimental setting, but also exhibit reasonable scalability. Our future work may consider more stochastic model other than normal distribution and/or randomizing other deterministic DAG scheduling heuristics.

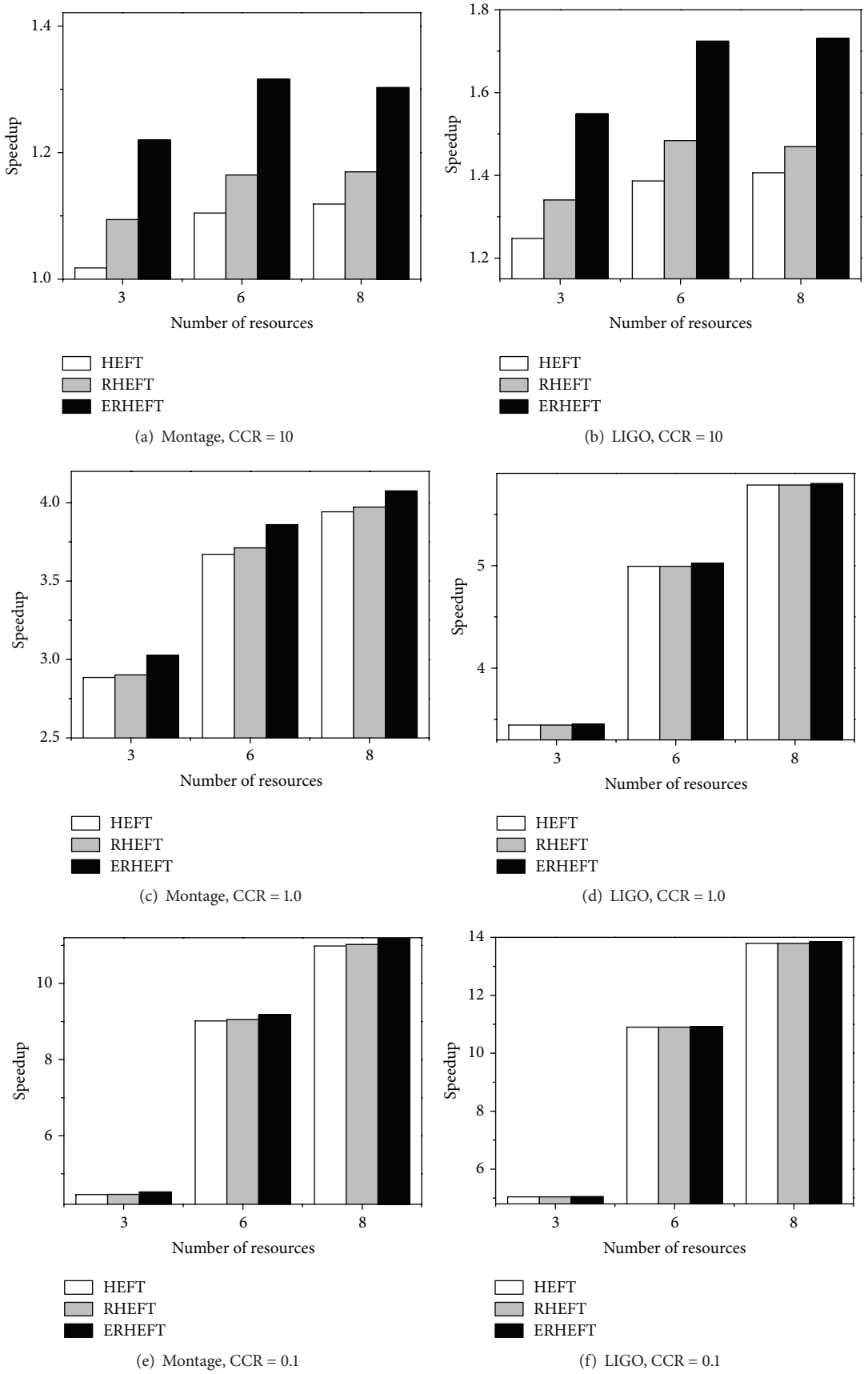


Figure 8: Average speedup results with different CCR values.

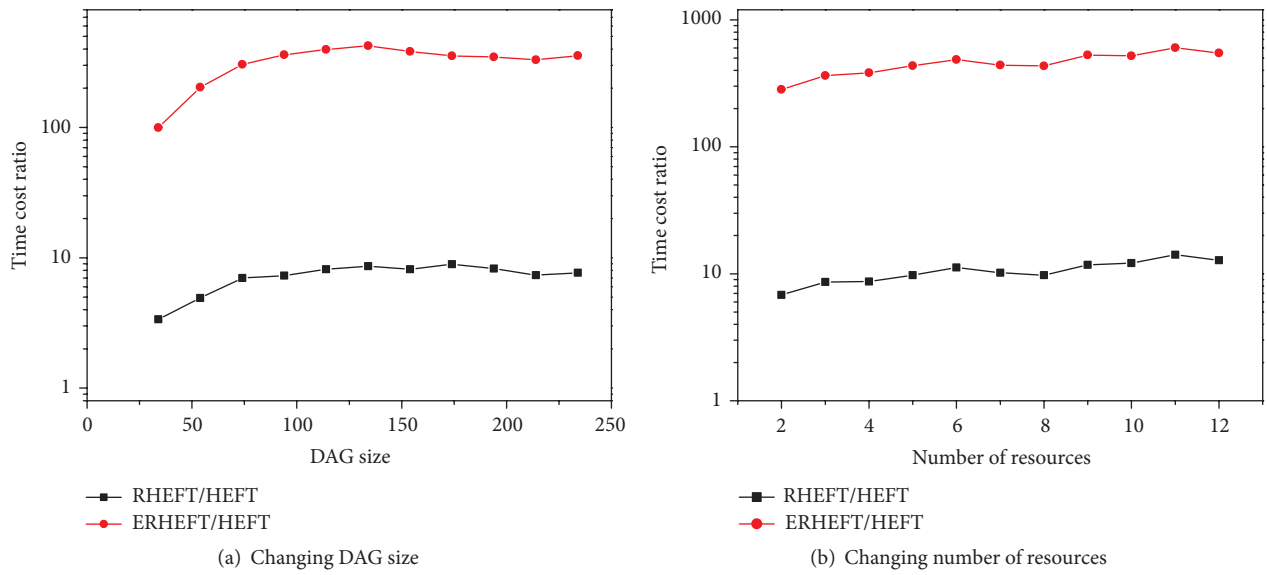


Figure 9: Time cost ratio of RHEFT and ERHEFT over HEFT.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The work is supported by National Natural Science Foundation of China (NSFC, Grant no. 61202361).

References

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: an overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[2] G. B. Berriman, J. C. Good, A. C. Laity et al., "A grid enabled image mosaic service for the national virtual observatory," in *Proceedings of the Conference Series of Astronomical Data Analysis Software and Systems XIII (ADASS XIII)*, pp. 593–596, 2004.

[3] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for E-Science: Scientific Workflows for Grids*, Springer, New York, NY, USA, 2007.

[4] H. Casanova, F. Dufossé, Y. Robert, and F. Vivien, "Scheduling parallel iterative applications on volatile resources," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS '11)*, pp. 1012–1023, IEEE, Anchorage, Alaska, USA, May 2011.

[5] S. Yeo and H. S. Lee, "Using mathematical modeling in provisioning a heterogeneous cloud computing environment," *IEEE Computer*, vol. 44, no. 8, pp. 55–62, 2011.

[6] G. Juve and E. Deelman, "Scientific workflows and clouds," *ACM Crossroads*, vol. 16, no. 3, pp. 14–18, 2010.

[7] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, "An evaluation of the cost and performance of scientific workflows on Amazon EC2," *Journal of Grid Computing*, vol. 10, no. 1, pp. 5–21, 2012.

[8] J. Li, D. Li, Y. Ye, and X. Lu, "Efficient multi-tenant virtual machine allocation in cloud data centers," *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 81–89, 2015.

[9] C. Cheng, J. Li, and Y. Wang, "An energy-saving task scheduling strategy based on vacation queuing theory in cloud computing," *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 28–39, 2015.

[10] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[11] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[12] R. Sakellariou and H. Zhao, "A hybrid heuristic for DAG scheduling on heterogeneous systems," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, IEEE Computer Society, Santa Fe , NM, USA, April 2004.

[13] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[14] M. L. Pinedo, *Overview of Stochastic Scheduling Problems*, Springer, Berlin, Germany, 2011.

[15] W. Zheng and R. Sakellariou, "Stochastic DAG scheduling using a Monte Carlo approach," *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1673–1689, 2013.

[16] R. Sakellariou and H. Zhao, "A hybrid heuristic for DAG scheduling on heterogeneous systems," in *Proceedings of the 13th Heterogeneous Computing Workshop*, pp. 111–124, 2004.

[17] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175–187, 1993.

[18] J. Blythe, S. Jain, E. Deelman et al., "Task scheduling strategies for workflow-based applications in grids," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, vol. 2, pp. 759–767, May 2005.

[19] G. Q. Liu, K. L. Poh, and M. Xie, "Iterative list scheduling for heterogeneous computing," *Journal of Parallel and Distributed Computing*, vol. 65, no. 5, pp. 654–664, 2005.

[20] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. MacIejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, vol. 47, no. 1, pp. 8–22, 1997.

[21] M. Coli and P. Palazzari, "Real time pipelined system design through simulated annealing," *Journal of Systems Architecture*, vol. 42, no. 6-7, pp. 465–475, 1996.

[22] B. Cirou and E. Jeannot, "Triplet: a clustering scheduling algorithm for heterogeneous systems," in *Proceedings of the International Conference on Parallel Processing Workshops*, pp. 231–236, Valencia, Spain, 2001.

[23] S. Ranaweera and D. P. Agrawal, "A task duplication based scheduling algorithm for heterogeneous systems," in *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, pp. 445–450, Cancun, Mexico, May 2000.

[24] S. Ranaweera and D. P. Agrawal, "A scalable task duplication based scheduling algorithm for heterogeneous systems," in *Proceedings of the International Conference on Parallel Processing*, pp. 383–390, Toronto, Canada, 2000.

[25] A. Dogan and R. Ozguner, "LDBS: a duplication based scheduling algorithm for heterogeneous computing systems," in *Proceedings of the International Conference on Parallel Processing (ICPP '02)*, pp. 352–359, IEEE, 2002.

[26] L. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative evaluation of the robustness of DAG scheduling heuristics," in *Grid Computing: Achievements and Prospects*, S. Gorlatch, P. Fragopoulou, and T. Priol, Eds., pp. 73–84, Springer, Berlin, Germany, 2008.

[27] S. A. Jarvis, L. He, D. P. Spooner, and G. R. Nudd, "The impact of predictive inaccuracies on execution scheduling," *Performance Evaluation*, vol. 60, no. 1–4, pp. 127–139, 2005.

[28] M. M. López, E. Heymann, and M. A. Senar, "Analysis of dynamic heuristics for workflow scheduling on grid systems," in *Proceedings of the 5th International Symposium on Parallel and Distributed Computing*, pp. 199–207, IEEE, Timisoara, Romania, July 2006.

[29] A. Kamthe and S.-Y. Lee, "A stochastic approach to estimating earliest start times of nodes for scheduling DAGs on heterogeneous distributed computing systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS '05)*, p. 121b, Denver, Colo, USA, April 2005.

[30] X. Tang, K. Li, G. Liao, K. Fang, and F. Wu, "A stochastic scheduling algorithm for precedence constrained tasks on grid," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1083–1091, 2011.

[31] K. Li, X. Tang, B. Veeravalli, and K. Li, "Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 191–204, 2015.

[32] W. Zheng, B. Emmanuel, and C. Wang, "A randomized heuristic for stochastic workflow scheduling on heterogeneous systems," in *Proceedings of the 3rd International Conference on Advanced Cloud and Big Data (CBD '15)*, pp. 88–95, Yangzhou, China, October 2015.

[33] E. Deelman, C. Kesselman, G. Mehta et al., "GriPhyN and LIGO, building a virtual data Grid for gravitational wave scientists," in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC '02)*, pp. 225–234, IEEE, 2002.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 7040276, 10 pages
http://dx.doi.org/10.1155/2016/7040276

Research Article

A Heuristic Task Scheduling Algorithm for Heterogeneous Virtual Clusters

Weiwei Lin,[1] Wentai Wu,[1] and James Z. Wang[2]

[1]*School of Computer Science and Engineering, South China University of Technology, Guangdong, China*
[2]*School of Computing, Clemson University, P.O. Box 340974, Clemson, SC 29634-0974, USA*

Correspondence should be addressed to Wentai Wu; cswuwt@mail.scut.edu.cn

Received 27 January 2016; Accepted 20 April 2016

Academic Editor: Ligang He

Cloud computing provides on-demand computing and storage services with high performance and high scalability. However, the rising energy consumption of cloud data centers has become a prominent problem. In this paper, we first introduce an energy-aware framework for task scheduling in virtual clusters. The framework consists of a task resource requirements prediction module, an energy estimate module, and a scheduler with a task buffer. Secondly, based on this framework, we propose a virtual machine power efficiency-aware greedy scheduling algorithm (VPEGS). As a heuristic algorithm, VPEGS estimates task energy by considering factors including task resource demands, VM power efficiency, and server workload before scheduling tasks in a greedy manner. We simulated a heterogeneous VM cluster and conducted experiment to evaluate the effectiveness of VPEGS. Simulation results show that VPEGS effectively reduced total energy consumption by more than 20% without producing large scheduling overheads. With the similar heuristic ideology, it outperformed Min-Min and RASA with respect to energy saving by about 29% and 28%, respectively.

1. Introduction

Cloud computing gains its popularity since it satisfies the elastic demands of computing capability from both individual and enterprise users. Cloud platforms not only support a diversity of applications, but also provide a virtualized environment for the applications to run in an efficient and low-cost manner [1]. As cloud computing is getting prevailing in IT industry, the huge amount of electricity consumed by cloud data centers also becomes a rising concern. According to the previous statistics, globally there are over 5 million data centers [2], which account for about 1.5% of the global energy consumption [3]. The figure may continuously go up as our demands for computing are still growing. Hence, in order to minimize the negative impact brought by energy wasting and overconsumption, it is of great necessity to improve resource utilization and to reduce energy consumption for cloud data centers.

Applying energy-aware resource scheduling is an effective way to save energy. Cloud data centers are usually virtualized. Thus in an IaaS (Infrastructure-as-a-Service) cloud, virtual machine (VM) is the basic unit for resource provisioning. After a user-defined job is submitted, it is first "sliced" into a number of tasks and generally each task will be assigned to one VM for execution. During the execution, the virtual resources allocated to the VM can be thought of being occupied by the task. The mapping from tasks to VMs is one-to-one. On the one hand, we do not consider a many-to-one mapping because resource competition often causes SLA (Service-Level Agreement) violations. On the other hand, one-to-many mapping can be avoided by a fine-grained job decomposition. Although the jobs or tasks may not contain any attributes initially, we can exploit available techniques to estimate their resources demands including total instructions, amount of disk I/O, and the data throughput on network. Besides, to attain the goal of saving task execution energy, it is of great necessity to consider servers' power efficiency. Assigning tasks to high-performance servers may enhance the data centers' overall performance but at the same time can cause extra energy

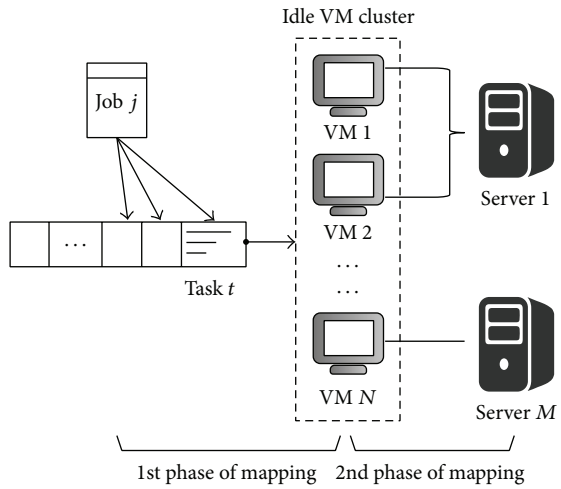


Figure 1: Resource scheduling in IaaS cloud.

consumption (i.e., operational cost). This is because some servers of high processing speed may not be power-efficient. Hence, we argue that the power efficiency of servers and VMs should be regarded as an important metric in today's energy-aware resource management.

Resource scheduling can be separated into two phases: task scheduling and VM scheduling. The first phase of mapping shown in Figure 1 represents task scheduling, which is the focus of this paper. Previous task scheduling algorithms (e.g., [4–6]) allocated tasks directly to physical servers. However, these algorithms are not rather feasible and effective since currently virtualization has been widely deployed on physical servers. The running environment for tasks is the virtual cluster. Besides, the majority of task scheduling algorithms use the strategy that VM is dynamically created on a selected server only when new tasks arrive. This kind of strategy is useful to aggregate workload in order to avoid too many idle servers. But it lowers the system's response ability as powering on a new VM takes time. An ideal target for task scheduling is to reduce system energy consumption with acceptable efficiency. Thus, in this paper, we propose to build a virtual cluster maintenance mechanism which combines VM "precreating" and "delayed shutdown." To be detailed, "precreating" means virtual machine can be started up on servers under relatively light workload before tasks arrive. "Delayed shutdown" allows a VM to stay alive for a certain period after it finishes its task. In cloud environment, this mechanism can maintain a large-scale idle VM cluster and thus allows a shorter task response time without bringing big overhead cost. At the same time, this mechanism is helpful to reduce migration operations, so it can be used to simplify VM consolidation (the 2nd mapping phase in Figure 1) strategies such as [7, 8].

Supported by VM "precreating" and "delayed shutdown" mechanism, we in this paper propose an energy-aware task scheduling framework for virtualized cloud environment. The framework consists of a task resource requirements prediction module, an energy estimate module, and a scheduler with a task buffer. The buffer works as an improvement on simple FIFO queue of arriving tasks. The size of the buffer is designed to be adaptive to the arrival rate of tasks. Receiving the output from the task resource requirements prediction module, task energy estimate module is responsible for estimating the energy consumption of executing. As the key part, the scheduler adopts a VM power efficiency-aware greedy scheduling algorithm (VPEGS) to schedule the tasks in the buffer heuristically. Experiments were conducted to evaluate the performance of VPEGS in a simulated heterogeneous virtual cluster. The results show that VPEGS averagely reduced more than 20% energy consumption and outperformed Min-Min [9], RASA [10], and Random-Mapping [11].

2. Related Work

Task scheduling has been proved to be a NP-problem [12]. Even with the mechanism of VM "precreating" and "delayed shutdown," task scheduling in a heterogeneous cloud is still a nontrivial problem. Heuristic scheduling algorithms such as Min-Min [9] and ant colony optimization [13, 14] are widely used in cloud task scheduling because they are quite efficient and sometimes able to approach optimal solutions [15]. Min-Min is a typical task scheduling algorithm oriented to heterogeneous infrastructures. Gutierrez-Garcia and Sim [11] compare 14 heuristic scheduling algorithms with respect to average task makespan. The results show that Min-Min and Max-Min [9] are the most effective among the algorithms using batch mode. Besides, Etminani and Naghibzadeh [16] proved that dynamically selecting Min-Min or Max-Min as the scheduler according to the standard deviation of expected task execution time can improve system performance. Priya and Subramani [10] propose a heuristic scheduling named RASA that consists of 3 phases. In initialization phase the execution efficiency matrix is initialized, while the scheduler finds the best-fit VM and returns its ID in the second and third phase. The idea of RASA is using Min-Min and Max-Min alternatively to schedule the tasks that arrived. Uddin et al. [17] tested and analyzed the performance of RASA, TPPC, and PALB in CloudSim considering power efficiency and cost as well as CO_2 emissions. They concluded that TPPC is most effective but neglected the detailed parameter settings of these algorithms.

Cloud servers are usually virtualized. Thus it is of great necessity to perform task scheduling in virtual clusters. Sampaio and Barbosa propose POFARE [4], considering both VM reliability and efficiency. This heuristic algorithm promotes the energy utilization (MFLOPS/Joules) but pays no attention to server virtualization. Lakra and Yadav [18] conducted task scheduling by solving a multiobjective optimization via nondominated sorting after quantifying the QoS values of tasks and VMs. However, it has the drawbacks of not being energy-aware and evaluating VM performance merely by MIPS (Million Instructions per Second). VM consolidation is another effective way to save energy with the basic ideology that powering off idle servers can reduce energy consumption. For example, HHCS [19], an energy-saving scheduling strategy, makes use of the advantages of two open-source schedulers (Condor and Haizea) in order

to further increase CPU utilization of physical servers. In addition, there are also implements (e.g., [20–22]) based on setting thresholds and constrains. Ding et al. [23] adopt this method to perform resource provisioning at the VM-level. Current technology allows dynamic VM migration, which is helpful to balance workload between servers. However, VM migration causes extra time and energy overheads. Hence, it is a better scheme to precreate and maintain a number of VMs on servers under light workload. Then these idle VMs can respond quickly when a new batch of tasks arrives.

3. Energy-Aware Task Scheduling Framework

3.1. Energy Estimate Module. In an IaaS cloud, virtualization makes physical resources "transparent" as the applications are run in VMs. To some extent, virtual machine provides independent runtime environment and it is also the basic unit allocated to user applications. In the proposed framework, the energy estimate module predicts the expected task energy consumption on each available VM and sends the data to the scheduler. For energy estimation, the required information includes task resource demands and the power efficiency of each VM.

Job submitted to the cloud will first be decomposed into several tasks. The decomposition principle can be data-based or function-based. Practically, total number of instructions and I/O data size can be estimated by analyzing the submitted code or exploiting other existing techniques. Actually there are many ways to estimate the resource demands of a task. The methods mentioned in [24] can be applied to process I/O-intensive tasks while, according to [25], the required amount of resources by the tasks belonging to the same job are usually similar. In this paper, we use four "static" attributes to profile a task: number of instructions, the size of data through disk input/output, the size of data through network transmission, and job_id indicating the job it is generated from. The values of these attributes remain unchanged despite the decisions of the scheduler. On the contrary, "dynamic" attributes, including the execution time and energy consumption of a task, are dependent on the features of the VM that executes it.

VM's power features are directly related to the features of its host. According to the definition of power efficiency, the power efficiency of a server can be defined in three aspects:

$$\begin{aligned} \mathrm{PE}_{\mathrm{proc}} &= \frac{\mathrm{proc\_perf}}{P_{\mathrm{proc}}}, \\ \mathrm{PE}_{\mathrm{io}} &= \frac{\mathrm{io\_rate}}{P_{\mathrm{io}}}, \\ \mathrm{PE}_{\mathrm{trans}} &= \frac{\mathrm{trans\_rate}}{P_{\mathrm{trans}}}, \end{aligned} \tag{1}$$

where proc_perf denotes the processor performance, which can be quantified using MIPS (Million Instructions per Second). io_rate and trans_rate represent the max disk I/O rate and max network transmission rate, respectively. Their metric is MB/s. P_{proc}, P_{io}, and P_{trans} are the power consumption of the corresponding functional components. All these data can be sampled on physical servers (e.g., we can obtain P_{proc} by measuring CPU power). It is worth noting that $\mathrm{PE}_{\mathrm{trans}}$ denotes the power efficiency in transport data between servers and it is stored in a matrix. $\mathrm{PE}_{\mathrm{trans}}$ are exploited to calculate the energy cost in multitask communications. In order to shield the complexity of network, we use a simple star network topology in designing the way that tasks communicate with each other, assuming that only tasks decomposed from the same job will conduct data transport between each other. We select one of them to be a "designated task" and other tasks follow the principle that they only send data to or receive data from the "designated task" (Figure 2).

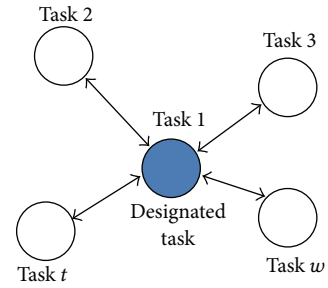


Figure 2: Simplified data exchanging by appointing designated task.

There exists a difference between the power efficiency of a VM and its host server because of virtualization. For example, different types of hypervisors suffer different degrees of degradation in VM performance. We use d_{proc}, d_{io}, and d_{trans} to represent the degradation in VM power efficiency of processing, disk I/O, and data transmission, respectively. Thus the power efficiency of a VM can be expressed as below:

$$\begin{aligned} \mathrm{PE}'_{\mathrm{proc}} &= \mathrm{PE}_{\mathrm{proc}} \cdot \left(1 - d_{\mathrm{proc}}\right), \\ \mathrm{PE}'_{\mathrm{io}} &= \mathrm{PE}_{\mathrm{io}} \cdot \left(1 - d_{\mathrm{io}}\right), \\ \mathrm{PE}'_{\mathrm{trans}} &= \mathrm{PE}_{\mathrm{trans}} \cdot \left(1 - d_{\mathrm{trans}}\right). \end{aligned} \tag{2}$$

As a summary, Table 1 lists the power features of VMs.

The dynamic power consumption of cloud data centers is mainly produced from the workload on each running server, while the resource demands of tasks are the major sources that drive server workloads. In cloud environment, the demands of tasks can be generally modeled by the task attributes mentioned above. However, it is very difficult to precisely predict the workload as a whole because actually a server has several components (e.g., CPU, memory, disk, and NIC) that keep producing static (idle) and dynamic power. Thus a possible way is to consider the workload of each component separately. We adopted this ideology and propose to calculate separately the power of computing, storage accessing, and communicating. Particularly in this paper we take the load of the whole server into account and use it to model performance loss.

Let P'_{proc}, P'_{io}, and P'_{trans} denote the VM's power consumption in processing, disk I/O, and network data transfer, respectively. We assume that VMs stay busy when executing the tasks assigned. So we regard P'_{proc}, P'_{io}, and P'_{trans}

Table 1: VM power features.

VM power features	Description
host_id	Indicates the VM's host server
proc_perf′	Max processing speed of the VM
io_rate′	Max disk I/O rate of the VM
trans_rate′	Max data transfer rate of the VM
PE'_{proc}	VM power efficiency (processor)
PE'_{io}	VM power efficiency (disk I/O)
PE'_{trans}	VM power efficiency (data transfer)
d_{proc}	PE_{cal} degradation
d_{io}	PE_{io} degradation
d_{trans}	PE_{trans} degradation

as unchanged values during execution. Considering task resource demands, VM power features, and the workload on host servers, we can estimate the energy consumption of a task run on a VM via

$$\begin{aligned} \text{task\_energy} &= E_{\text{proc}} + E_{\text{io}} + E_{\text{trans}} \\ &= P'_{\text{proc}} \cdot T_{\text{proc}} + P'_{\text{io}} \cdot T_{\text{io}} + P'_{\text{trans}} \cdot T_{\text{trans}}, \end{aligned} \tag{3}$$

where

$$\begin{aligned} T_{\text{proc}} &= \frac{N_{\text{proc}}}{\text{proc\_perf}' \cdot (1 - L)}, \\ T_{\text{io}} &= \frac{N_{\text{io}}}{\text{io\_rate}' \cdot (1 - L)}, \\ T_{\text{trans}} &= \frac{N_{\text{trans}}}{\text{trans\_rate}' \cdot (1 - L)}, \end{aligned} \tag{4}$$

where N_{proc} denotes the number of instructions, while N_{io} and N_{trans} represent the amount of disk data throughput and the amount of data transferred through network, respectively. These task attributes can be estimated by existing techniques. L is the performance loss caused by high workload on the server. It is intuitive that the higher the load a server works under, the greater value L has. The correlation between the workload of CPU and other components is quite complex, but there is a basic knowledge that the performance of the whole system probably degrades when CPU is working under high load. So as a simplification, we model L as follows:

$$L = \begin{cases} u^{1/\beta}, & 0 \le u \le 0.95, \\ 1, & 0.95 < u < 1, \end{cases} \tag{5}$$

where u represents the current CPU utilization of the host server. β is the high-load penalty factor and $\beta \in (0, 1]$. With (3) and (4), we finally have

$$\text{task\_energy} = \left(\frac{N_{\text{proc}}}{PE'_{\text{proc}}} + \frac{N_{\text{io}}}{PE'_{\text{io}}} + \frac{N_{\text{trans}}}{PE'_{\text{trans}}} \right) \cdot \frac{1}{1 - L}, \tag{6}$$

where PE'_{proc}, PE'_{io}, and PE'_{trans} represent the power efficiency of VM regarding instructions processing, disk I/O, and data transmission. From (6) we can see that assigning tasks to virtual machines with high power efficiency is of great significance to reduce energy consumption. Meanwhile, the workload on servers should also be considered because high load leads to great performance degradation, which increases the energy required to finish a task.

3.2. Task Buffer. There are two methods to determine the scheduling order: FIFO mode and buffer mode (or batch mode). In completely FIFO mode, all tasks are organized and scheduled sequentially according to the arrival time. Thus FIFO mode provides best fairness but may fail to satisfy the QoS (Quality of Service) of some specific tasks. As an improvement, buffer mode allows buffering a certain number of tasks and schedules them by some principles. Buffer mode is similar to priority queue but it is not global, which guarantees the scheduler's efficiency and enhances its effectiveness at the same time. Algorithms that adopt buffer or batch mode include Min-Min, Max-Min [9], and RASA [10]. Practically, it is not easy to determine the buffer's size because oversized buffer causes low efficiency while making it too small may reduce the chance to find better scheduling solutions.

In this paper, a variable-sized task buffer is adopted on the basis of a global FIFO queue. To be more detailed, tasks at the head of the FIFO queue are put in the buffer then their minimum energy consumption (relevant to currently available VMs) will be estimated. Tasks with lower predicted energy consumption will be scheduled with higher priorities. Assume that the arriving of cloud tasks is a Poisson process with its intensity equal to λ; then the expectation of task arrival interval is $1/\lambda$. Hence, it is a feasible way to set the buffer size to a multiple of λ (and round it):

$$\text{buf\_size} = \lceil \alpha \cdot \lambda \rceil, \tag{7}$$

where α is a system parameter that can be set empirically. Increasing the size of buffer is helpful to find better (more energy-saving) scheduling solutions when tasks arrive intensively. Meanwhile a smaller buffer can make the scheduler more efficient in the condition that the arrival rate is relatively low.

3.3. Task Scheduling Framework. Now we briefly depict the entire energy-aware task scheduling framework. After being submitted to the cloud, users' jobs are first decomposed into several tasks. These tasks are put in a FIFO queue and then those at the head are transferred to the task buffer. The energy estimate module is in charge of estimating the energy consumption of each task in the buffer. After receiving the output from energy estimate module, the scheduler finishes the scheduling of this batch of tasks. Then the next batch is pushed into the buffer and the above process is repeated. Figure 3 illustrates the whole energy-aware task scheduling framework.

The algorithm inside the scheduler is the key part for making energy-saving task allocations. Thus we propose an energy-saving heuristic task scheduling algorithm.

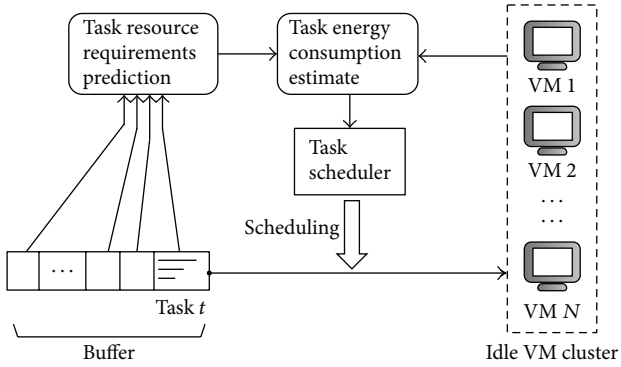


Figure 3: Energy-aware task scheduling framework.

Table 2: List of parameters used in the algorithm's pseudocode.

Parameter	Description
V	The set of currently available VMs
M	The set of physical servers
Buffer	Task buffer
Q	Global task queue (FIFO)
buf_size	The size of task buffer

4. VM Power Efficiency-Aware Greedy Scheduling Algorithm (VPEGS)

With the expansion of cloud data centers and the increase of computing demands from users, it is of great significance to consider the heterogeneity of both infrastructures and task demands. Currently, many researches (e.g., [23, 26]) only cast their sight on VM consolidation because it is an effective way to reduce wasted energy by controlling the workload on servers. However, if much load is imposed on servers with low power efficiency, it will cause higher energy cost to warrant the QoS of tasks, which is the situation that service providers are unwilling to face.

A feasible and effective solution is to consider power efficiency in task scheduling. In virtualized environment, colocated VMs can be regarded to have equal power efficiency, which can be calculated by applying (2). Thus, assuming that the infrastructure supports VM precreating and delayed shutdown, we propose a virtual machine power efficiency-aware greedy scheduling algorithm (VPEGS). The algorithm takes VM power efficiency and task demands into account and provides a sort of energy-saving task scheduling. We first list the parameters used in the algorithm and give brief descriptions (Table 2).

VPEGS is heuristic and takes the estimated task execution energy as the evaluation function. We exploit (6) to estimate the execution energy consumption ($\text{task\_energy}_{t,k}$) of task t on VM k, considering VM efficiency, efficiency loss caused by virtualization, and the performance loss caused by high server workload. Since we adopt task buffer, the process of scheduling is similar to Min-Min and RASA. In other words, the program attempt to search the buffer for a (t^*, k^*) satisfies

$$\text{task\_energy}_{t^*,k^*} = \min\left\{\text{task\_energy}_{t,k}\right\}, \quad (8)$$

```
Input: V, M, Q
Output: task-to-VM Mapping
(1)  Initialize Buffer
(2)  Initialize min_energy = MAX_FLOAT
(3)  while Q is not empty do
(4)    for i = 1 to min{size(Q), buf_size} do
(5)      t = dequeue(Q)
(6)      add t into Buffer
(7)    end
(8)    while Buffer is not empty do
(9)      for each task t in Buffer do
(10)       for each VM k in V do
(11)         calculate task_energy_{t,k}
(12)         if task_energy_{t,k} < min_energy then
(13)           min_energy = task_energy_{t,k}
(14)           selected_task = t
(15)           selected_VM = k
(16)         end if
(17)       end for
(18)     end for
(19)     assign selected_task to selected_VM
(20)     remove task t from Buffer
(21)     update the states of V and M
(22)   end while
(23) end while
(24) return task-to-VM Mapping
```

Algorithm 1: Virtual machine power efficiency-aware greedy scheduling algorithm (VPEGS).

where $t = 0, 1, \ldots, (\text{buf\_size} - 1)$ and $k = 0, 1, \ldots, n$. n is the number of VMs currently available. Then in this round, the scheduler assigns task t to VM k. The pseudocode of VPEGS is shown in Algorithm 1.

The task buffer is initialized first and then the global FIFO queue which dequeues the tasks at the head. After the buffer is filled (or FIFO queue becomes empty), the scheduler computes $\text{task\_energy}_{t,k}$ for each task t on every available VM k (line (11)). Its time complexity equals inspecting a $\text{buf\_size} * n$ sized matrix. The minimum element is found and the corresponding VM ID and task ID are recorded (lines (14)~(15)). Then the selected task is assigned to the selected VM. This process repeats until the buffer is clear. Then the next batch of tasks will be sent into it.

We analyze the complexity of VPEGS as below: each decision of assignment has to check the whole matrix whose size is $\text{buf\_size} * n$, so the complexity of assigning one task is $O(\text{buf\_size} * n)$. Suppose the total number of tasks that arrived is u. Thus the overall time complexity of finishing the scheduling is $O(u * \text{buf\_size} * n)$.

5. Algorithm Evaluation

5.1. Experimental Setup. We implemented VPEGS and evaluated it in a simulated environment. We also implemented Min-Min [9], RASA [10], and Random-Mapping [11] in order to compare their effectiveness. The algorithms and test programs were written in Java (JDK version 1.8.0_65).

Table 3: The setting of task attributes in the experiment.

Task attribute	Value
N_{proc}	$\sim U(170, 510)$
N_{io}	$\sim U(100, 3000)$
N_{trans}	$\sim U(100, 2000)$
Threads	{1, 2, 3, 4}

Table 4: The setting of server configurations in the experiment.

Type	PE_{proc}	PE_{io}	d_{proc}	d_{io}	d_{trans}	Number
Server 0	0.33	10.50	0.2	0.2	0.1	20
Server 1	0.15	9.00	0.1	0.3	0.1	8
Server 2	0.17	9.50	0.1	0.2	0.1	20
Server 3	0.28	21.50	0.1	0.1	0.1	12
Server 4	0.21	15.50	0.2	0.2	0.1	40

The simulation was run on a PC equipped with a dual-core Pentium CPU (2.10 GHz) and 4.0 GB memory.

For every task decomposed from a job, the experimental setting of its attributes is listed in Table 3. Where the metric of N_{proc} is Million instructions, while the unit of N_{io} and N_{trans} is MB. In order to simulate the difference of power efficiency between heterogeneous physical servers, we set 5 types of servers and the corresponding configurations are listed in Table 4. In the experiment, we suppose the power efficiency of data transfer is infinity (i.e., zero overhead) if two VMs are server-local. Otherwise, it equals 50. Meanwhile, the task with the smallest task_id is always appointed to be the "designated task" when multiple tasks that belong to the same job are active in the virtual cluster. Thus the elements in PE_{trans} matrix are defined as

$$\text{PE}_{\text{trans}}(i, j) = \begin{cases} +\infty, & i = j, \\ 50, & i \neq j. \end{cases} \tag{9}$$

5.2. Experimental Results. In the experiment we set the number of servers (m) to 100 according to Table 4 fixedly. The program randomly generated 250 to 300 VMs in the initialization phase. High-load penalty factor β was set to 0.15. The intervals of task arrivals followed exponential distribution with $\lambda = 3$ and the buffer size was set to 15 initially. After initialization, the test program utilized VPEGS, Min-Min, RASA, and Random-Mapping (RM) separately as the scheduling strategy to run the simulation. Each test repeated 30 times and we took the average as our results. The comparison regarding total system energy consumption is shown in Figure 4.

The result illustrates that VPEGS performed the best among the four scheduling algorithms with respect to energy saving (Figure 4). Min-Min and RASA had similar performance since their heuristic principles behind are similar. VPEGS saved 29.1% and 28.6% energy when compared to them on average. As for the reason, we argue that Min-Min and RASA in some way can be energy-saving because shortening overall execution time reduces the consumption

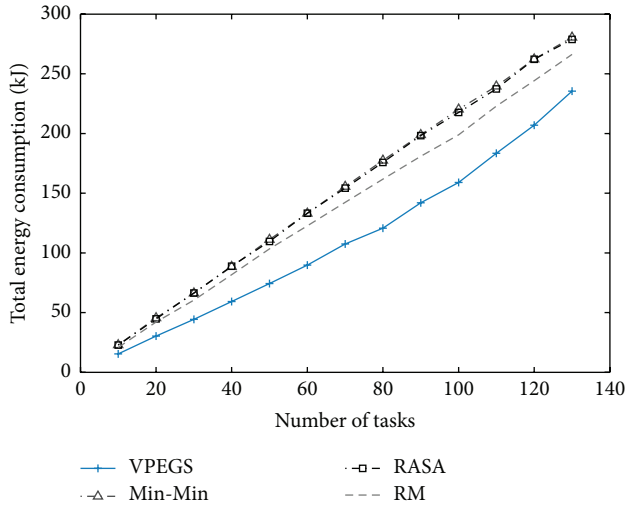


Figure 4: Comparing the performances of VPEGS, Min-Min, RASA, and RM on total energy consumption (buffer size = 15).

brought by server idle power. However, as power efficiency is not taken into account, assigning tasks to those high-performance nodes may cause extra energy consumption. On the contrast, VPEGS considers both the performance and power features of VMs and exploits power efficiency as the prime metric. Specifically speaking, Min-Min and RASA are more likely to utilize the servers with great processing speed or throughput rate, whereas VPEGS prefers those with high power efficiency. In our experiment, actually a big number of high-performance servers were of comparatively low power efficiency. As a result, VPEGS showed its advantage in energy saving. It is also noticed that Random-Mapping (RM) seemed to be slightly more energy-saving than Min-Min and RASA. Essentially this is because RM assigns tasks evenly so usually high workload would not be imposed on servers with low power efficiency. Averagely, VPEGS outperforms Random-Mapping by about 23.0%.

We also see that when the number of servers is fixed, it seems to be tougher to maintain energy-saving performance as the number of tasks increases (Figure 5). When virtual resources are sufficient to satisfying tasks' demands, using VPEGS can reduce total energy consumption by more than 20%. However, as the task arrival rate remained unchanged ($\lambda = 3$), the workload of the whole cluster went high as the total number of tasks increased. In other words, comparatively energy-efficient VMs were gradually used up. Onto the fixed-scale simulated data centers with 100 heterogeneous servers, the performance of VPEGS degraded in our experiment when the number of tasks is more than 90 (Figure 5).

We mentioned that the size of task buffer may influence the performances of scheduling algorithms. To verify it, we changed the task arrival rate, namely, λ, to 6. Correspondingly, the size of buffer was adjusted to 30. We reinitialized the clusters and conducted the experiment again. Figures 6 and 7 show the results. In this case, compared with Min-Min, RASA, and RM, VPEGS averagely saved 29.8%, 29.0%, and

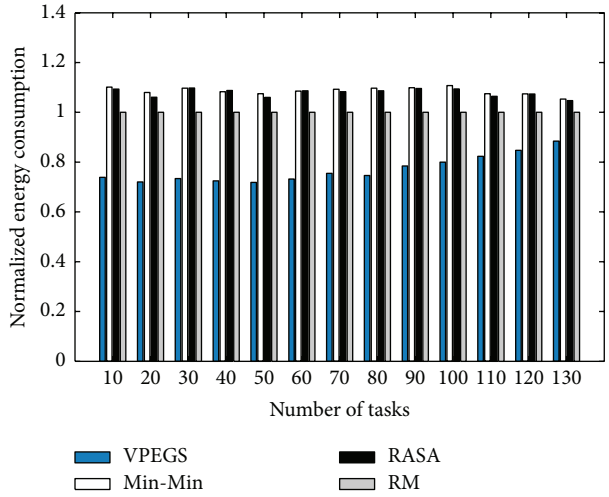


Figure 5: Normalized energy consumptions with different total number of tasks (buffer size = 15).

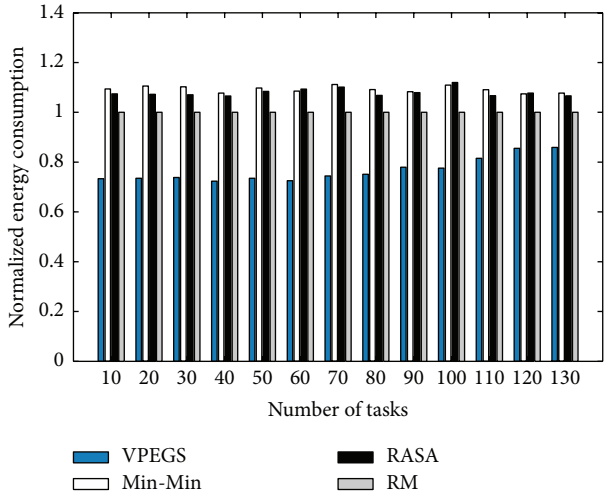


Figure 7: Normalized energy consumptions with different total number of tasks (buffer size = 30).

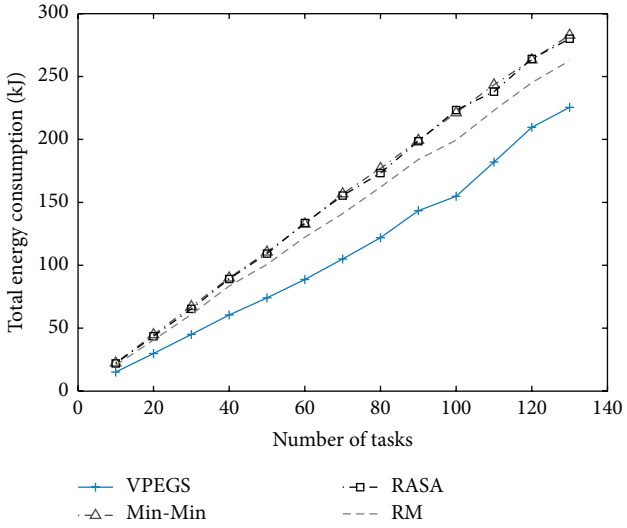


Figure 6: Comparing the performances of VPEGS, Min-Min, RASA, and RM on total energy consumption (buffer size = 30).

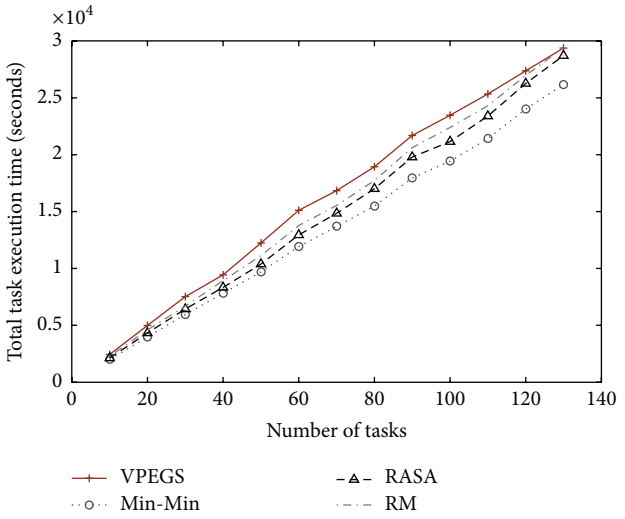


Figure 8: Comparing different task scheduling algorithms on the total task execution time (buffer size = 15).

23.3% energy, respectively. It is a little surprising to find that enlarging the task buffer does not have big impact. But we also see that the performance of VPEGS in this case was slightly improved when the number of tasks exceeded 100.

Tasks may not gain their earliest completion time in VPEGS since energy consumption is considered primarily. There is a kind of conflict, as mentioned in [27], between optimizing execution time and energy consumption. VPEGS, to some degree, sacrifices the efficiency of task execution to attain the goal of saving more energy. However, Min-Min and RASA pay more attention to reducing total makespan and task execution time. Figure 8 shows the total task execution time of Min-Min, RASA, Random-Mapping, and VPEGS with the buffer size equal to 15. As a result, Min-Min and RASA are effective in shortening the overall execution time of all the tasks. The reason is simple: Min-Min and RASA take predicted task completion time as the heuristic. Besides, short tasks outnumbered long tasks in our experiment; thus RASA yielded no better performance in reducing total execution time than pure Min-Min. We ran the test again after changing the task arrival rate and the buffer's size (Figure 9). Combining Figure 8 and Figure 9, we can see that the change of buffer size did not affect the total task execution time for VPEGS. But the time for RASA was shortened when we reduced the number of tasks per batch (Figure 9). This is because when the number of tasks in the buffer was reduced, RASA was more likely to make the same decisions with Min-Min.

We also carried out experiments to test the impact of the buffer's size on scheduling overheads (Figure 10). Scheduling overhead represents the average time that the scheduler takes to make a task assignment decision. We use the average time

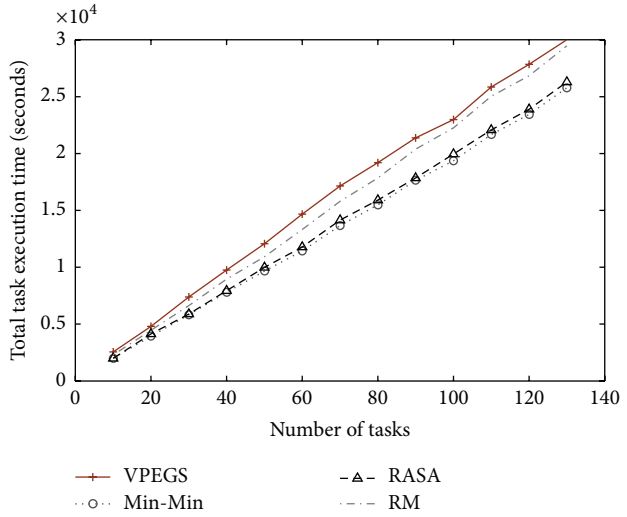


Figure 9: Comparing different task scheduling algorithms on the total task execution time (buffer size = 5).

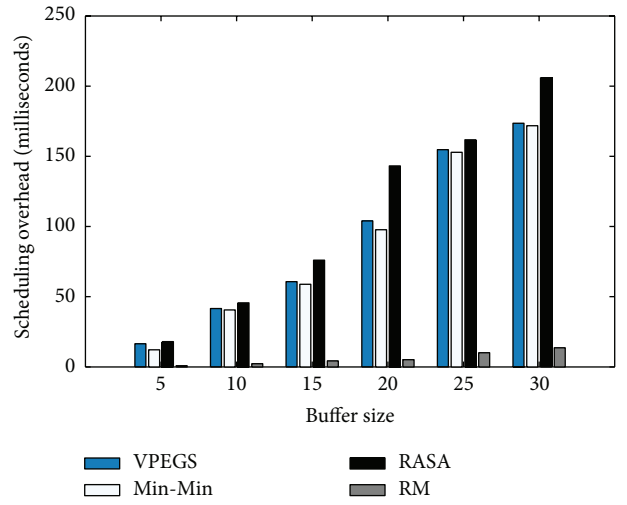


Figure 10: The scheduling overheads of VPEGS, Min-Min, RASA, and RM with different buffer sizes.

that a task stays in the buffer to evaluate the scheduler's efficiency in the experiment.

Though theoretically Min-Min, RASA as well as VPEGS have the same time complexity; it can be pointed out from Figure 10 that Min-Min suffers the least scheduling overheads among these three heuristic algorithms. The reason is that VPEGS spends extra time on estimating task energy. RASA checks whether the number of available VMs is odd before assigning tasks. VPEGS is slightly more efficient than RASA since it does not check the odd-even property and only considers current availability of VMs. In other words, tasks will not wait for occupied VMs in VPEGS even though sometimes waiting helps to shorten the makespan and execution time. On this point, we conclude that VPEGS, as a heuristic algorithm, only suffers small scheduling overheads when adopting an appropriate size of task buffer.

As a summary, the experimental results illustrate that as the scheduler of our proposed energy-aware scheduling framework, VPEGS is effective to schedule tasks in an energy-saving manner. Compared with traditional scheduling algorithms focusing on optimizing overall makespan and task execution time, VPEGS takes into account the task resource demands, VM power efficiency, and server workload. The target of algorithms like Min-Min and RASA are to shorten the makespan and total execution time of a batch of tasks. This is in some way helpful to save system energy when the differences between servers' power efficiencies are small and server idle power makes great impact on the total energy consumption. However, with the fast expanding on data centers' scale, cloud infrastructures probably consist of hundreds of different types of servers. This heterogeneity makes it necessary to consider more factors including server performance, power efficiency, and server workloads. Aiming at reducing the energy consumption of heterogeneous clusters, VPEGS provides a highly feasible way to conduct energy-aware task scheduling. We list the main advantages of VPEGS as follows:

(i) Multiple factors that influence system energy consumption are considered. VPEGS conducts scheduling according to the estimation on task energy, which takes into account the information about task resource demands, VM power efficiency, server workload, and performance loss.

(ii) VPEGS realizes a fine-grained resource provisioning and task scheduling at the level of virtual machine clusters supporting "precreating" and "delayed shutdown."

(iii) VPEGS has high feasibility since it works without any training. Besides, we set the size of the task buffer to an adaptive value to balance the scheduler's performance and efficiency.

(iv) Greedy strategy is made use of to realize low-overhead task scheduling.

6. Conclusion and Future Work

Cloud computing is believed to have great potential in satisfying diverse computing demand from both individuals and enterprises. But at the same time the overconsumption of electricity by cloud data centers becomes a big worry. Considering the virtualized environment in cloud data centers, in this paper, we propose an energy-aware task scheduling framework consisting of a task resource requirements prediction module, an energy estimate module, and a scheduler. Based on this framework, we propose a heuristic task scheduling algorithm named VPEGS. VPEGS takes into account task resource demands, server/VM power efficiency as well as server workload. Oriented to heterogeneous cloud environment, the proposed algorithm does not need training and is able to schedule tasks in an energy-saving manner. VPEGS shares the similar heuristic ideology with Min-Min and RASA, but it prominently saves system energy by sacrificing some efficiency in task execution. Experiment

based on simulation was carried out to evaluate VPEGS. The results illustrate its novelty that VPEGS reduced system energy consumption by over 20% when compared to the strategy of Random-Mapping. It also outperformed Min-Min and RASA in saving energy by approximately 29% and 28%, respectively, without producing large scheduling overheads.

Future research will focus on how to effectively combine task scheduling and VM consolidation strategies in order to further enhance the effectiveness of energy saving. Besides, we plan to make a deeper investigation into the factors or technologies (e.g., Dynamic Voltage Frequency Scaling) that influence server's power efficiency.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

Acknowledgments

Thanks are due to the helpful comments and suggestions from the anonymous reviewers. This work is partially supported by the National Natural Science Foundation of China (Grant no. 61402183), Guangdong Natural Science Foundation (Grant no. S2012030006242), Guangdong Provincial Scientific and Technological Projects (Grants nos. 2016A010101007, 2016B090918021, 2014B010117001, 2014A010103022, 2014A010103008, 2013B090200021, and 2013B010202001), Guangzhou Science and Technology Projects (Grants nos. 201601010047 and 20150504050525159), and Fundamental Research Funds for the Central Universities, SCUT (no. 2015ZZ0098).

References

[1] M. Pedram, "Energy-efficient datacenters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 10, pp. 1465–1484, 2012.

[2] How many data centers?, March 2013, http://www.datacenterknowledge.com/archives/2011/12/14/how-many-data-centers-emerson-says-500000/.

[3] J. Koomey, "Growth in data center electricity use 2005 to 2010," A Report by Analytical Press, 2011.

[4] A. M. Sampaio and J. G. Barbosa, "Towards high-available and energy-efficient virtual computing environments in the cloud," *Future Generation Computer Systems*, vol. 40, pp. 30–43, 2014.

[5] Z. Deng, G. Zeng, Q. He, Y. Zhong, and W. Wang, "Using priced timed automaton to analyse the energy consumption in cloud computing environment," *Cluster Computing*, vol. 17, no. 4, pp. 1295–1307, 2014.

[6] Y. Tian, C. Lin, and K. Li, "Managing performance and power consumption tradeoff for multiple heterogeneous servers in cloud computing," *Cluster Computing*, vol. 17, no. 3, pp. 943–955, 2014.

[7] H. M. Lee, Y.-S. Jeong, and H. J. Jang, "Performance analysis based resource allocation for green cloud computing," *The Journal of Supercomputing*, vol. 69, no. 3, pp. 1013–1026, 2014.

[8] A. Horri, M. S. Mozafari, and G. Dastghaibyfard, "Novel resource allocation algorithms to performance and energy efficiency in cloud computing," *Journal of Supercomputing*, vol. 69, no. 3, pp. 1445–1461, 2014.

[9] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel & Distributed Computing*, vol. 59, no. 2, pp. 107–131, 1999.

[10] S. M. Priya and B. Subramani, "A new approach for load balancing cloud computing," *International Journal of Engineering and Computer Science*, vol. 2, no. 5, pp. 1636–1640, 2013.

[11] J. O. Gutierrez-Garcia and K. M. Sim, "A family of heuristics for agent-based elastic Cloud bag-of-tasks concurrent scheduling," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1682–1699, 2013.

[12] D. Fernández-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, 1989.

[13] L. Zhu, Q. Li, and L. He, "Study on cloud computing resource scheduling strategy based on the ant colony optimization algorithm," *International Journal of Computer Science Issues*, vol. 9, no. 5, pp. 54–58, 2012.

[14] E. Pacini, C. Mateos, and C. G. Garino, "Balancing throughput and response time in online scientific clouds via ant colony optimization (sp2013/2013/00006)," *Advances in Engineering Software*, vol. 84, pp. 31–47, 2015.

[15] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 275–295, 2015.

[16] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorithm for grid task scheduling," in *Proceedings of the 3rd IEEE/IFIP International Conference in Central Asia on Internet (ICI '07)*, pp. 1–7, Tashkent, Uzbekistan, September 2007.

[17] M. Uddin, Y. Darabidarabkhani, A. Shah, and J. Memon, "Evaluating power efficient algorithms for efficiency and carbon emissions in cloud data centers: a review," *Renewable & Sustainable Energy Reviews*, vol. 51, pp. 1553–1563, 2015.

[18] A. V. Lakra and D. K. Yadav, "Multi-objective tasks scheduling algorithm for cloud computing throughput optimization," *Procedia Computer Science*, vol. 48, pp. 107–113, 2015.

[19] H. Kurdi and E. T. Alotaibi, "A hybrid approach for scheduling virtual machines in private clouds," *Procedia Computer Science*, vol. 34, pp. 249–256, 2014.

[20] W. Lin, J. Z. Wang, C. Liang, and D. Qi, "A threshold-based dynamic resource allocation scheme for cloud computing," *Procedia Engineering*, vol. 23, pp. 695–703, 2011.

[21] W. Lin, B. Liu, L. Zhu, and D. Qi, "CSP-based resource allocation model and algorithms for energy-efficient cloud computing," *Journal on Communications*, vol. 34, no. 12, pp. 33–41, 2013 (Chinese).

[22] W. Lin, C. Liang, J. Z. Wang, and R. Buyya, "Bandwidth-aware divisible task scheduling for cloud computing," *Software: Practice and Experience*, vol. 44, no. 2, pp. 163–174, 2014.

[23] Y. Ding, X. Qin, L. Liu, and T. Wang, "Energy efficient scheduling of virtual machines in cloud with deadline constraint," *Future Generation Computer Systems*, vol. 50, pp. 62–74, 2015.

[24] Q. Zhao, C. Xiong, C. Yu, C. Zhang, and X. Zhao, "A new energy-aware task scheduling method for data-intensive applications in the cloud," *Journal of Network and Computer Applications*, vol. 59, pp. 14–27, 2016.

[25] X. Bu, J. Rao, and C. Z. Xu, "Interference and locality-aware task scheduling for MapReduce applications in virtual clusters,"

in *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '13)*, pp. 227–238, ACM, New York, NY, USA, June 2013.

[26] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[27] Y. Wang and X. Wang, "Performance-controlled server consolidation for virtualized data centers with multi-tier applications," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 1, pp. 52–65, 2014.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 1808396, 12 pages
http://dx.doi.org/10.1155/2016/1808396

Research Article

MHDFS: A Memory-Based Hadoop Framework for Large Data Storage

Aibo Song,[1] Maoxian Zhao,[2] Yingying Xue,[1] and Junzhou Luo[1]

[1]*School of Computer Science and Engineering, Southeast University, Nanjing 211189, China*
[2]*College of Mathematics and Systems Science, Shandong University of Science and Technology, Qingdao 266590, China*

Correspondence should be addressed to Aibo Song; absong@seu.edu.cn

Received 22 February 2016; Accepted 17 April 2016

Academic Editor: Laurence T. Yang

Hadoop distributed file system (HDFS) is undoubtedly the most popular framework for storing and processing large amount of data on clusters of machines. Although a plethora of practices have been proposed for improving the processing efficiency and resource utilization, traditional HDFS still suffers from the overhead of disk-based low throughput and I/O rate. In this paper, we attempt to address this problem by developing a memory-based Hadoop framework called MHDFS. Firstly, a strategy for allocating and configuring reasonable memory resources for MHDFS is designed and RAMFS is utilized to develop the framework. Then, we propose a new method to handle the data replacement to disk when memory resource is excessively occupied. An algorithm for estimating and updating the replacement is designed based on the metrics of file heat. Finally, substantial experiments are conducted which demonstrate the effectiveness of MHDFS and its advantage against conventional HDFS.

1. Introduction

Recent years have seen an astounding growth of enterprises having urgent requirements to collect, store, and analyze enormous data for analyzing important information. These requirements continuously step over a wide spectrum of application domains, ranging from e-business and search engine to social networking [1, 2].

With the significant increment of data consumption in various applications, from the level of GB to PB, there is an urgent need for such platforms with superior ability to store and process this exploding information. As the most used commercial computing model based on the Internet, cloud computing is currently employed as the primary solution and can provide reliable, scalable, and flexible computing abilities as well as theoretically unlimited storage resources [3]. With necessary components available, such as networks, computational nodes, and storage media, large-scale distributed clouding platforms can be conveniently and quickly built for various data-intensive applications.

Conventional could computing systems mainly rely on distributed disk storages and employ the management subsystems to integrate enormous machines and build the effective computing platform. Typical cloud computing platforms include Apache Hadoop [4], Microsoft Azure [5], and Google MapReduce [6, 7], among which the open-source Hadoop system gains particular interests in practice. Generally, these platforms all provide high throughput in data access and data storing for clients by effectively managing distributed computer resources, which are proved to be appropriate to store and process large amount of data in real-world applications.

However, as the evolvement of modern computer hardware, the capacity of various memory media is continuously increasing, with the decline of their prices. Considering this situation, researchers and engineers have focused their effort on the employment and management of memory resources to step across the bottleneck of I/O rate in conventional distributed cloud computing systems.

Current memory-based distributed file systems are mostly designed for the real-time applications, including HANA [8] and Spark [9, 10] as the typical representatives. Their goal is to speed up the process of data writing and reading from different perspectives. HANA is the platform

developed by SAP Inc. and is widely used in fast data processing by employing a specifically designed framework of memory-based distributed storing. Even though HANA platform is widely adopted, it is mainly designed to handle the structured data instead of semistructured data. Since there are quite a number of applications based on semistructured or even unstructured data, HANA is not suitable for them. Meanwhile, HANA is usually memory-intensive due to its high occupation of memory space. Spark is a recently involved cloud computing system based on memory that employs the basic concept of resilient distributed datasets (RDD) to effectively manage the data transformation and storage. Spark is a fast and general-purpose cluster computing system which provides high-level APIs and an optimized engine that supports general job executions. It is highly dependent on the third-party component Mesos [11], whose authority is to manage and isolate various memory resources. This causes the unreliability and absence of customization for Spark. Additionally, Spark is based on its self-developed data model called RDD (resilient distributed datasets) that differs significantly from the conventional HDFS data model. Thus, the problem of compatibility is also the crucial part for its limited employment on existing applications. Developers have to design their data structures and architectures from scratch.

Based on the above analysis, this paper mainly focuses on improving the current infrastructure of HDFS. We extended the storage media of HDFS from solely disk-based to memory-based, with disk storage as addition, and designed a proper strategy to preferentially store data into the memory. Meanwhile, we proposed an algorithm of memory data replacement for handling the overload of limited memory space and effectively increase the average I/O rate as well as overall throughput of the system.

The main difference of our MHDFS framework compared to Spark and HANA includes two aspects. Firstly, we developed the framework based on native Hadoop, which provides consistent APIs and data structures. So existing applications based on Hadoop can conveniently migrate to MHDFS with little changes. Secondly, we designed the memory data replacement module as the secondary storage. So MHDFS can automatically handle the situation when memory space is nearly occupied.

The remainder of this paper is organized as follows. Section 2 presents a brief description of our proposed MHDFS system and demonstrates the key architecture. Section 3 gives the strategies of allocating and deploying distributed memory resources. Section 4 introduces fault tolerance design based on memory data replacement. Then, in Section 5, experiments on different size of data are conducted to evaluate the effectiveness of our proposed model. Finally, we present related studies in Section 6 and conclude our work in Section 7.

2. Architecture

Based on the previous analysis, we now present MHDFS, a memory-based Hadoop framework for large data storage. MHDFS is an incremental system of the native HDFS. Other than the normal modules, it includes two other modules, the memory resource deployment module and the memory data replacement module. The architecture of MHDFS is shown in Figure 1. In MHDFS, name node accepts clients' requests and forwards them to assigned data nodes for specific writing or reading jobs. Data nodes are based on both memory and disk spaces, where memory space is selected as the preferred storing media against disk through the help of memory resource deployment module. Meanwhile, for handling the problem of limited memory space, memory data replacement module is involved as the middleware and swaps files into disk when necessary. And it is proved to be useful to the robustness of the whole system.

2.1. Memory Resource Deployment Module. This module is designed for configuring and allocating available memory space for deploying MHDFS. Based on the analysis on the historical physical memory usage of each data node, we estimate the size of available memory space, that is, the remaining memory space not occupied by ordinary executions of a machine. Then, the estimated memory space will be mapped to a file path and used for deploying MHDFS. Data will be preferentially written into the memory space instead of the conventional HDFS disk blocks.

2.2. Memory Data Replacement Module. This module is designed for swapping data between memory and disk when RAM space is occupied by executing jobs. In this module, a reasonable upper bound is set to start the data replacement process. In addition, a submodel based on file accessed heat is proposed to find the best swapping data block. Intuitively, the file heat is related to the access frequency of a file; that is, files being accessed frequently during recent period will generally have higher heats. Details will be presented in Section 3. In this submodel, heat of each file will be evaluated and files in the memory will be ranked according to their heat. Then, files with low heat will be swapped out to the local disk so as to ensure the available memory space for subsequent job executions.

3. Strategy of Memory Resource Deployment

3.1. Allocation of Memory Resources. In order to allocate available memory resources for deploying HDFS, we utilized the records of physical memory on each data node and estimate the reasonable memory spaces. The key strategy is stated as follows.

We denote the total memory capacity of each data node as M_T, which is related to the configuration of the physical machine. To calculate the available memory space for deploying MHDFS, we monitor the usage information of data node within a time duration of ΔT and record the maximal RAM usage as M_h. Since the memory usage is fluctuating continuously with enormous job executions, we give a relax to the local computing usage and set the maximal usage as $\delta \times M_h$. The value of δ is determined by the running status of different clusters, but generally we could set it as 120% to satisfy the local job executing requirements. Finally,

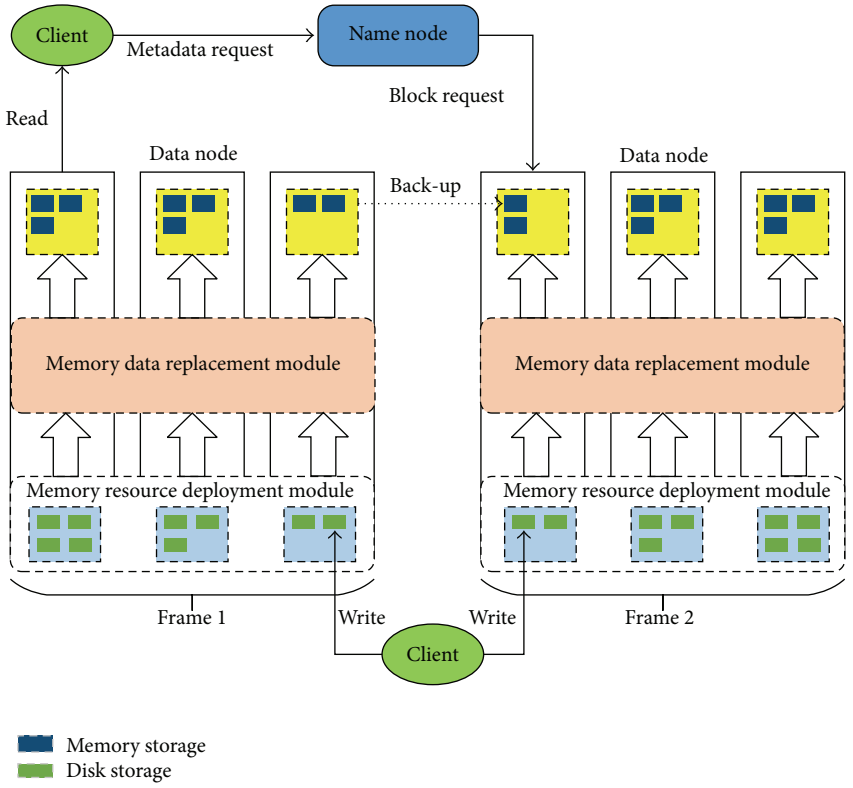


Figure 1: The architecture of MHDFS.

the remaining memory space is allocated as the resources for MHDFS, which is called the maximum available memory storage space (M_a):

$$M_a = M_T - M_h * 120\%. \quad (1)$$

If the calculated M_a is less than or equal to zero, the historical maximal memory usage on the node is convinced to exceed 83.3%, which further indicates the heavy load of physical memory. In this case, due to the lack of enough available memory spaces, this data node is decided to be unsuitable for allocating memory resources for MHDFS. Intuitively, when the total size of written data exceeds M_a, MHDFS will prevent subsequent data writing into the memory of this node.

Since HDFS requires the storage paths mapping to specific directories, we have to map the available memory space to a file path. For the sake of convenience, *Ramfs* is used as the developing tool [12]. It is a RAM-based file system integrated into the Linux kernel and works at the layer of virtual file system (VFS).

According to the maximal available memory space M_a proposed above, we use *Ramfs* to mount the memory space to a predefined file path, so that data could be written into the memory accordingly. *Ramfs* takes the advantage of dynamically allocating and utilizing memory resources. To be more specific, if the amount of written data is less then M_a, say M_d ($M_d < M_a$), *Ramfs* only occupies the memory space of size M_d while the remaining are still preserved for ordinary job executions. With the size of written data increasing continuously, *Ramfs* enlarges the total occupied memory space to meet system requirements. The size of the remaining memory space could be marked as

$$M_p = M_T - M_d, \quad (2)$$

where M_T is the total memory space of the data node and M_d is the actual used memory space by MHDFS.

Ramfs ensures M_p always being positive along with the process of data reading and writing, which improves the effectiveness of memory utilization of the physical machine. And the mechanism of dynamically allocating memory resources further reduces its inferior effect to the capability of local job executions. Practically, we can set parameters in *hdfs-site.xml* file and configure the file path mapped from memory to another storage path of data block.

3.2. Management of Memory Resource. Conventionally, data is usually stored in the unit called data block in HDFS. Traditional data storing process in HDFS could be summarized as the following steps: (1) client requests for writing data, (2) name node takes charge of assigning data nodes for writing and recording the information of assigned nodes, and (3) specific data nodes store the writing data to its self-decided file paths.

The data storage structure of the data node could be demonstrated as in Figure 2. Each FSVolume represents a data storage path and the FSVolumeSet manages all FSVolumes which is configured in the directory of ${*dfs.data.dir*}. Generally, there is more than one data storage path ${*dfs.data.dir*} on each data node; HDFS employs

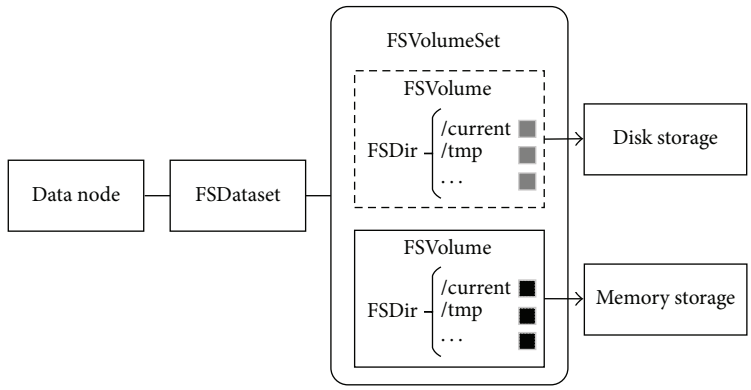


Figure 2: The data storage structure of MHDFS.

the strategy of round robin to select different paths so that data at each path could be balanced. The general idea of round-robin path selection is to use variable *curVolume* to record current selected storage path increasing it by one after a block of data is written each time. Then *curVolume* points to the next storage path in the volumes array. When each data block is written, storage path is changed and finally, after *curVolume* reaches the end of volumes array, it will be redirected to the head.

To ensure data written into the memory preferentially, we rewrite the policy of selecting storage path in HDFS. We assign different file paths with different priorities, sort them ordered by priority, and then store the file paths into the *volumes* array of the data node. Considering Figure 2, for example, with the memory-based storage path in array volumes[0] and the disk-based storage path in array volumes[1], we check the paths in the array from start when data is written. MHDFS will continuously check the paths subsequently until a satisfied one is found out.

4. Strategy of Memory Data Replacement

4.1. File Replacement Threshold Setting. Obviously, memory resource of physical machines is limited compared with disk spaces. When the remaining memory space of the data node is below a specific threshold, the process of replacing data stored in memory into disk will be triggered. In MHDFS, we design a strategy of threshold measuring and replacement trigger.

The threshold setting is specific to the user and application, whereas it is usually acceptable to set as the 10% of the maximal available storage space of the data node, which means when there is less than 10% of maximal available space remaining, MHDFS should start the process of replacement.

Based on the assumption that the maximum available memory on a data node DN is M_a and the already used memory space on DN is M_u, the trigger condition could be set that the remaining memory space is less than 10% of M_a. The data replacement trigger threshold on the DN node is denoted as T_{start}, and its formula could be wrote as follows:

$$T_{\text{start}} = (M_a - M_u) - 10\% * M_a. \tag{3}$$

After the writing operation has finished on DN, the detection of the threshold T_{start} will be conducted.

Replacing process is constantly conducted with low heat file swapped out from the memory until (4) is satisfied, which indicates that the available memory space on DN is sufficient again:

$$T_{\text{stop\_s}} \leq \frac{M_a - M_u}{M_a}. \tag{4}$$

Generally, the threshold for stopping the replacement process is very specific to different applications and configurations of clusters, which is decided by the user himself. In this paper, according to the environment of our experimenting cluster, the default stopping threshold is set as $T_{\text{stop\_s}} = 40\%$; that is, when the available memory on DN exceeds $40\% * M_a$ after the involvement of memory data replacement, the replacement process will be suspended and data will be still kept retaining in memory to ensure the efficiency of subsequent data accesses.

4.2. Measurement of File Heat. In order to make the file replacement effective, a strategy has to be proposed to identify the files to be replaced. Intuitively, if a file is accessed frequently by the applications, it is more preferable to retain in memory, while the less frequently visited files could be swapped into the disk to save memory space.

A frequency-based heat parameter is proposed in MHDFS to handle the problem stated above. The heat parameter will be stored in the metainformation of each file in the memory, which records the access frequency and access volume. Basically, there are three rules:

(1) Files with more frequent access have higher heat.

(2) Files with larger amount of data access have higher heat.

(3) Files with shorter survival time have higher heat.

We designed the heat metric based on the theory of time series which can evaluate the accessing status of each file. The frequency of the file being accessed is recorded within time interval. The heat is updated after the file is accessed. The

Input:
Operation: Type of Operate of Write and Read
RDatNum: Number of byte of requesting to access the file, or Number off byte of writing into file
⟨*Fseq, Heat, Life, AccessT, InVisitT*⟩: File List, Current Heat, Life Time, Access Time, Access Interval
Output: Result: Result of Update
(1) **Function** UpdateHeatDegree
(2) Seq = Fseq;
(3) //Get current time;
(4) Time = getTime();
(5) **if** (operation == read) **then**
(6) factor = (Time − Access$T$)/Invisit$T$ + (Time − Access$T$);
(7) New_heat = (factor ∗ RDatNum + (1 − factor) ∗ heat)/(Time − Life);
(8) Invisit$T$ = Time − Access$T$;
(9) Access$T$ = Time;
(10) Heat = New_Heat;
(11) **return** Success, ⟨Fseq, Heat, Life, Access$T$, InVisit$T$⟩
(12) **end if**
(13) **if** (operation == write) **then**
(14) Access$T$ = Time;
(15) Invisit$T$ = Time − Access$T$;
(16) **if** (operation == create) **then**
(17) Fseq = new Fseq();
(18) New_heat = 0;
(19) Life = Time;
(20) Heat = New_heat;
(21) **return** Success, ⟨Fseq, Heat, Life, Access$T$, InVisit$T$⟩
(22) **end if**
(23) **if** (operation == add) **then**
(24) **return** Success, ⟨Fseq, Heat, Life, Access$T$, ReSet$T$⟩
(25) **end if**
(26) **end if**
(27) **if** (operation == delete) **then**
(28) delete(Fseq);
(29) **return** null
(30) **end if**
(31) **return** Fail, ⟨Fseq, Heat, Life, Access$T$, ReSet$T$⟩
(32) **End**

Algorithm 1: Memory file heat update.

shorter the interval is, the greater impact the access has, and vice versa.

We denote $H_i(f)$ as the head of the file on data nodes in the memory storage and propose the following equation:

$$H_0(f) = 0$$
$$H_i(f) = \frac{\beta * S_i + (1-\beta) * H_{i-1}(f)}{T_i}, \tag{5}$$

where $\beta = \Delta t_i/(\Delta t_i + \Delta t_{i-1})$.

Apparently, the equation is consistent with the above three principles. When a file is accessed frequently, $H_{i-1}(f)$ gives more weight to $H_i(f)$, and if bytes of accessed data S_i are large or file survival time T_i is small, $H_i(f)$ will also generate a larger result. β is an adjustment factor based on the time series, which mainly works for adjusting the relationship between the current heat and historical heat. A larger adjustment factor results in a greater impact of data accessed amount to the file heat, while a smaller one gives more weight to the historical heat value.

4.3. File Heat Update. As can be seen from Section 4.2, file heat updating process is specific to different data operations. In this paper, we categorize various data operations into three groups: (1) writing operation, including creating and adding, (2) reading operation, and (3) deleting operation. Considering the three kinds of operations, the updating strategy is demonstrated as pseudocodes in Algorithm 1.

The updating of file heat information includes adding, updating, and deleting. Initially, the heat of a file is zero which clearly means that newly created file with no reading access never has a positive heat. And if a file has never been read since creation, the associated heat value will remain unchanged. This indicates that reading operation is closely

related to the increase of file heat which is confirmed to our common sense. The updating process can be summarized as follows.

(1) For reading operations, accessed data size and accessing time will be recorded, and our algorithm will update the file heat as well as the access interval parameter according to (5).

(2) For writing operations, we consider them as two cases: (a) creating operation: a file sequence number and file feat are initialized, with both creating time and access time as current system time and access interval as zero; (b) adding operation: adding time is recorded and access interval is updated, without updating the file heat. This is consistent with the above analysis that only reading operations can directly impact the altering of file heat.

(3) For deleting operations, file heat record will be deleted based on the defined file sequence number.

4.4. File Replacement. Based on the measurements of threshold setting and file heat updating strategy stated above, we designed the module of memory file replacement. The process can be summarized in Figure 3.

For the sake of simplicity, we set the starting threshold of replacement as 10% of total available memory space and the stopping threshold as 40% of total available memory space. The process could be summarized as the following steps:

(1) Check the remaining memory, and if remaining memory space is less than 10% of M_a, replacement process shall be started.

(2) Sort all files in memory by their file heats in descending order.

(3) Generally, each file may contain a number of data blocks. Replacement process starts from the data block with the longest survival time until all the data blocks are moved out.

(4) When all data blocks in a file have been replaced into the disk, the corresponding file heat is deleted and replacement process carries on to replace the data block in the file with secondary lowest file heat.

(5) Suspend replacement process when the remaining space reaches 40% of M_a again.

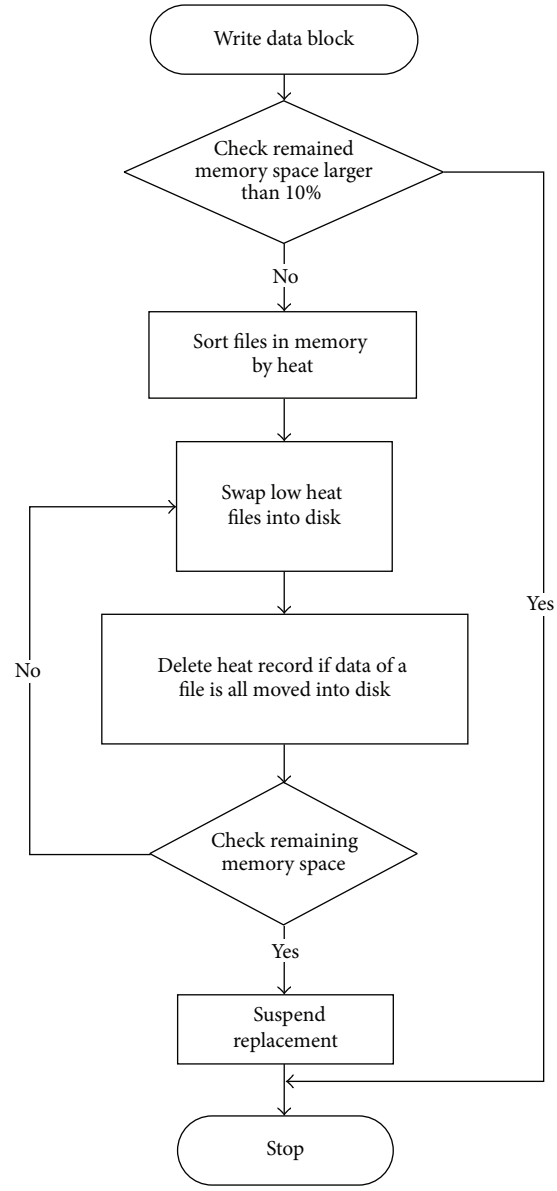


Figure 3: Process of memory file replacement.

5. Experiment

In order to showcase the performance of our proposed MHDFS model, we did comparison experiments on stimulated datasets. Reading and writing performance is evaluated on different size of data, say, 2 GB and 8 GB. *Ramfs* is utilized as the developing tool for the mapping of memory space.

5.1. Experimental Setup. Experimental cluster is set up with nine nodes: one for the Hadoop name node and resource manager, and the others for the data nodes and data managers. Each node is a physical machine with dual 3.2 GHz Xeon EM64 CPU, 4 GB MEM, and 80 GB HDD, connected via Gigabit Ethernet. We use Red Hat Enterprise Linux Server Release 6.2 as the host operating system and the version of Hadoop distribution is 2.6.0.

For conducting the experiments, we map the memory of name node to the file storage path using *Ramfs* and configure two file paths (one for the disk storage and the other for the memory storage) in the Hadoop configuration file. After recompilation, HDFS can identify both the file paths and dynamically assign priority to them.

5.2. Performance Comparison on Single-Node Cluster

5.2.1. I/O Test with Small Datasets (Less Than 2 GB). Data reading and writing operations are conducted on MHDFS

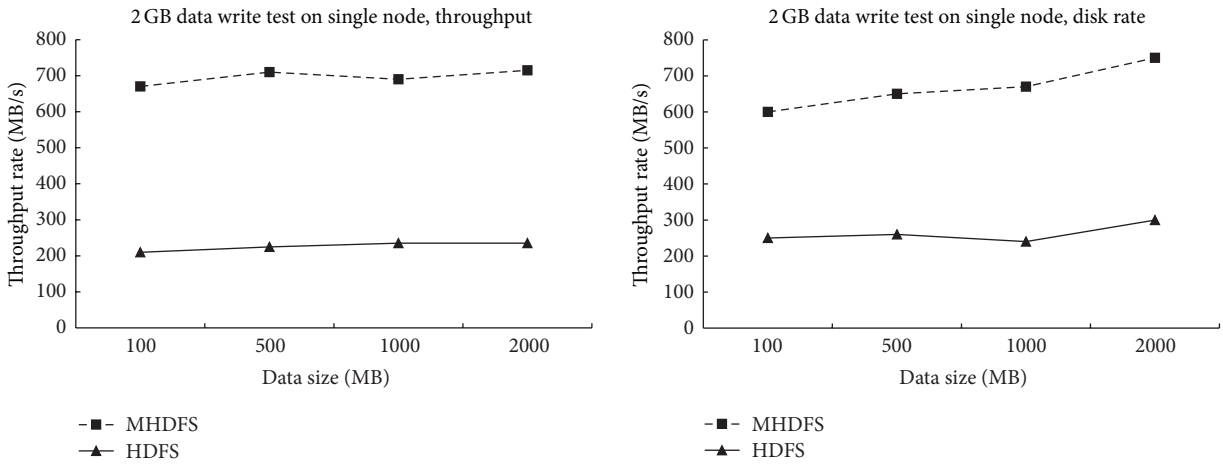


Figure 4: HDFS/MHDFS 2 GB data write test on single-node cluster.

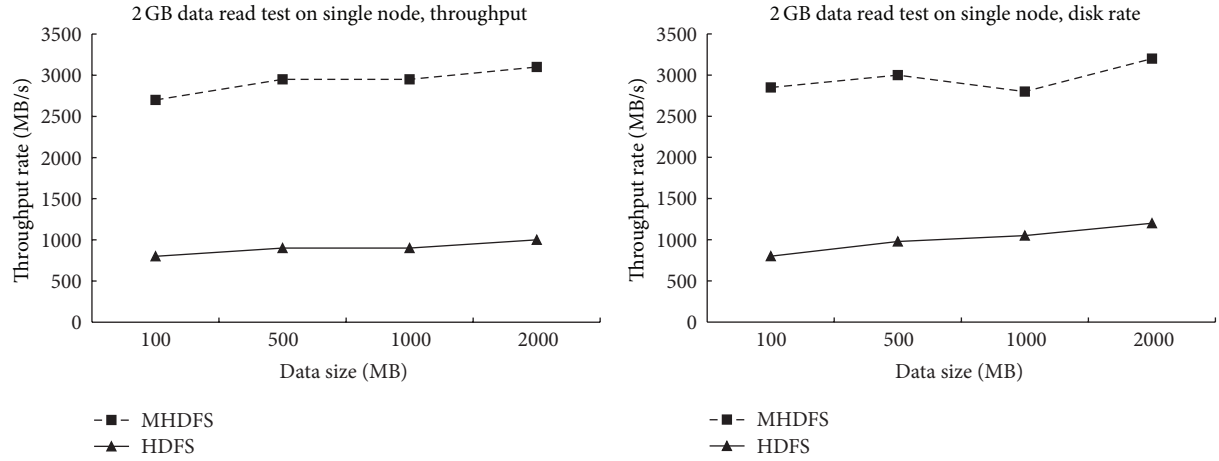


Figure 5: HDFS/MHDFS 2 GB data read test on single-node cluster.

and HDFS based on a single-node cluster and performance is recorded to evaluate the effectiveness of our proposed model. Data size increases continuously from 500 MB to 2 GB. The evaluating results are shown in Figures 4 and 5.

Generally, the throughput and average disk write rate of both models increase with the increment of data size. The throughput of HDFS is floating around 300 MB/s, whereas MHDFS reaches up to 700–800 MB/s, almost three times faster than HDFS. The result shows that MHDFS can significantly improve the performance of writing data compared with conventional HDFS.

Considering the reading performance shown in Figure 5, the throughput of HDFS is floating around 900 MB/s, while MHDFS is up to 3000 MB/s, which is also three times of HDFS. Internally, MHDFS is able to respond to data reading requests in time, since the data read by MHDFS is preferentially stored in memory, rather than disk. Besides, the relatively good performance of HDFS is highly related to the use of SSD, which further indicates the significant improvement of MHDFS against HDFS in various conditions.

5.2.2. I/O Test with Large Datasets (from 2 GB to 8 GB). In this subsection, we evaluate our proposed MHDFS model with larger size of datasets. Intuitively, when the written data exceeds a certain size, memory space is occupied, and subsequent writing will be directed to file paths in the disk.

Figure 6 shows the results of performance evaluation with data size varying from 2 GB to 8 GB. Generally, after the data size reaches over 2 GB, the throughout and average I/O rate both have declined due to the lack of memory space. In detail, the value of throughput and average I/O rate of writing and reading have declined from 800 Mps to 600 Mps and 3000 Mps to 2000 Mps, respectively. Even though both MHDFS and HDFS get declined performance with data increases, MHDFS still achieves higher results against HDFS. This is consistent with our expectation and can be explained as follows. When memory space is insufficient, data replacement operation will be started and data can only be stored into disk during the replacing period. This may lead to the declined performance of MHDFS for a while. After the complete of replacement, data is still written into the available

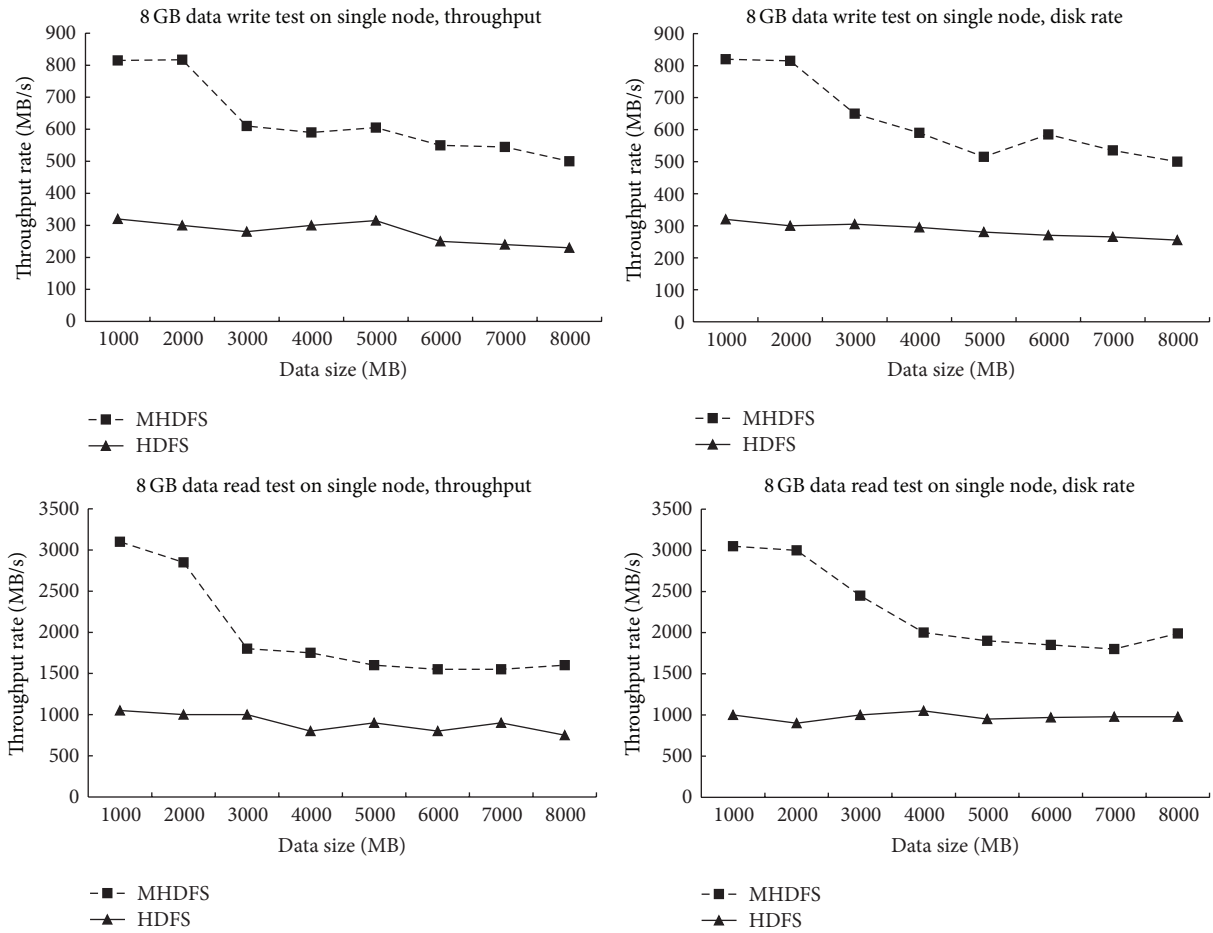


Figure 6: HDFS/MHDFS 8 GB data read/write test on single-node cluster.

memory space and reaches a relatively stable performance with subsequent writing requests.

5.3. Performance Comparison on Multinode Cluster. For comparing the performance of MHDFS and conventional HDFS in real-world distributed environment, we conducted experiments on the multinode cluster. We configure different memory resources for MHDFS on each node to test the triggering and swapping strategy. The configuration is shown in Table 1.

Generally, nine nodes are selected for the experiment: one for master node and the others for data nodes. The total memory allocated for MHDFS is 10100 MB, where master node is allocated 2 GB for ensuring the execution of the whole cluster. Memory allocation on each data node is according to the historical maximal memory usage individually. We use the Linux command *top* to periodically monitor the ordinary memory usage of each machine and estimate the maximal one. And for testing the effectiveness of our memory data replacement strategy, the experiment data size is designed to exceed the available total memory space.

5.3.1. I/O Test with Small Datasets (Less Than 10 GB). We evaluate the performance of MHDFS and HDFS on small datasets, say, less than 10 GB, when memory space for the whole cluster is not fully occupied. In this case, MHDFS will not trigger the process of data swapping and disk storing. The amount of data is increasing from 1 GB to 10 GB, every 2 GB as the step. The results are shown in Figures 7 and 8.

Table 1: Memory storage resources on each node.

Node	Historical maximal memory usage	Allocated memory size (MB)
Master	/	2000
$N1$	59.7%	1250
$N2$	62.6%	1000
$N3$	65.0%	900
$N4$	61.9%	1050
$N5$	68.1%	750
$N6$	58.9%	1200
$N7$	62.7%	1000
$N8$	64.1%	950

As can be seen from Figure 7, the writing throughput and average writing rate of MHDFS are 45 Mbps and 50 Mbps, respectively, which only has a slight improvement over HDFS

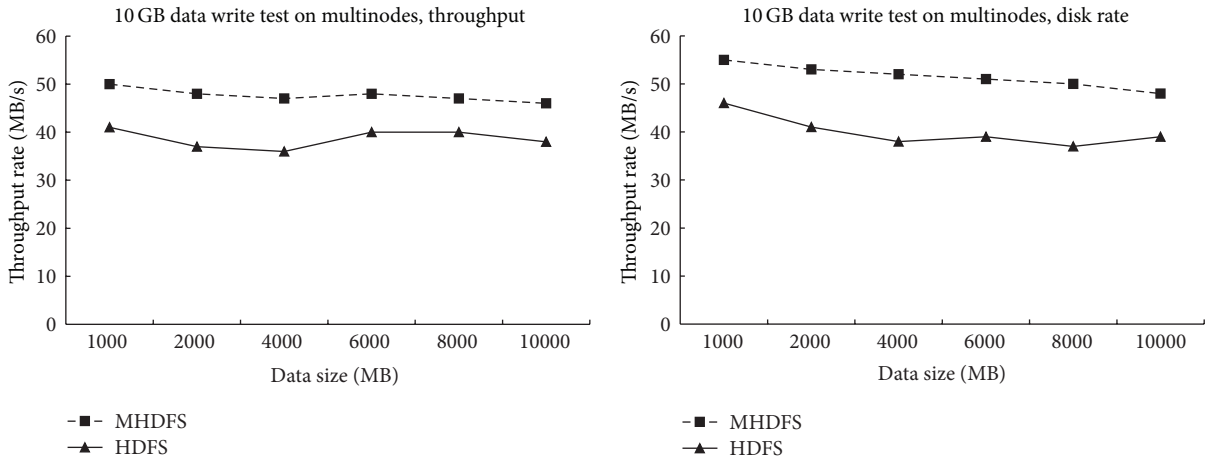


Figure 7: HDFS/MHDFS 10 GB data write test on multinode cluster.

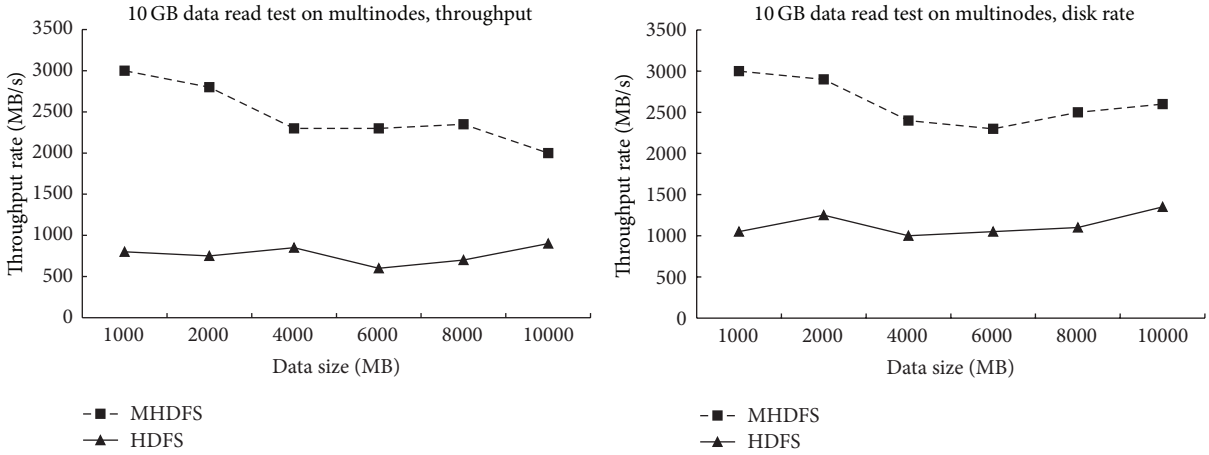


Figure 8: HDFS/MHDFS 10 GB data read test on multinode cluster.

compared with results in single-node environment. This is due to the fact that writing data into the cluster requires simultaneously copy writing; that is, for each writing, three copies of the data should be stored in different nodes. With 100 Mbps as the bandwidth of each node, the crucial cost of writing is actually the network overhead which cannot be improved by MHDFS. In fact, MHDFS is designed to leverage the memory space for handling disk writing overhead and thus can provide little help to other factors like network transferring.

Figure 8 presents the results of reading test. The reading throughput and average reading rate of MHDFS reaches 2600 Mbps and 2650 Mbps, respectively, which outperforms two and half times compared to HDFS. Undoubtedly, this is consistent with our theory that when reading data from cluster, data as well as its copies are mostly stored in memory and this provides very fast reading operations. Interestingly, however, with the increase of data size, the reading performance gets a few declines. This is because, for the single-node environment, all data can be read from memory while, for multinode clusters, some data must be transferred via network from other data nodes. The additional cost of network transfer leads to the slightly decline when large amount of data is requested.

5.3.2. I/O Test with Large Datasets (from 10 GB to 25 GB). In this subsection, we evaluate our proposed MHDFS model with larger size of datasets in the environment of multinodes cluster. In this situation, MHDFS will employ the strategy of data replacement. Figures 9 and 10 demonstrate the writing and reading results.

From Figure 9, it is clear that when writing data exceeds 10 GB, that is, the allocated memory space for MHDFS, the writing performance declines due to the involvement of data replacement, even if it still outperforms HDFS with 5% to 15%.

Remarkably, different with the cases in single-node cluster, the adjective impact of data replacement is much slighter due to the fact that, in single-node situation, memory space is totally prohibited for writing when data replacement is conducted, while, for multinode cluster, replacing on one node will not affect another node's writing process; that is, data replacement seldom occurs on all nodes in the whole cluster.

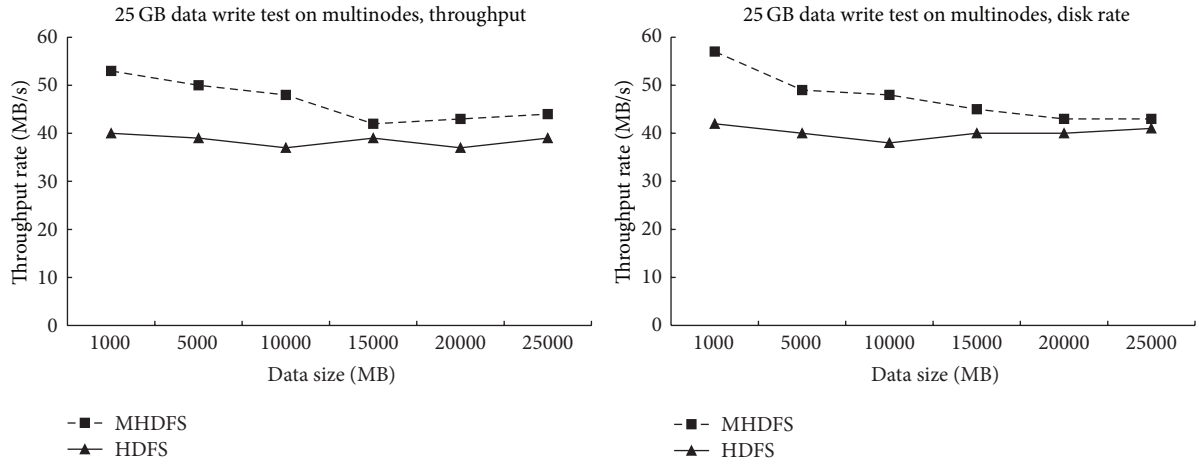


Figure 9: HDFS/MHDFS 25 GB data write test on multinode cluster.

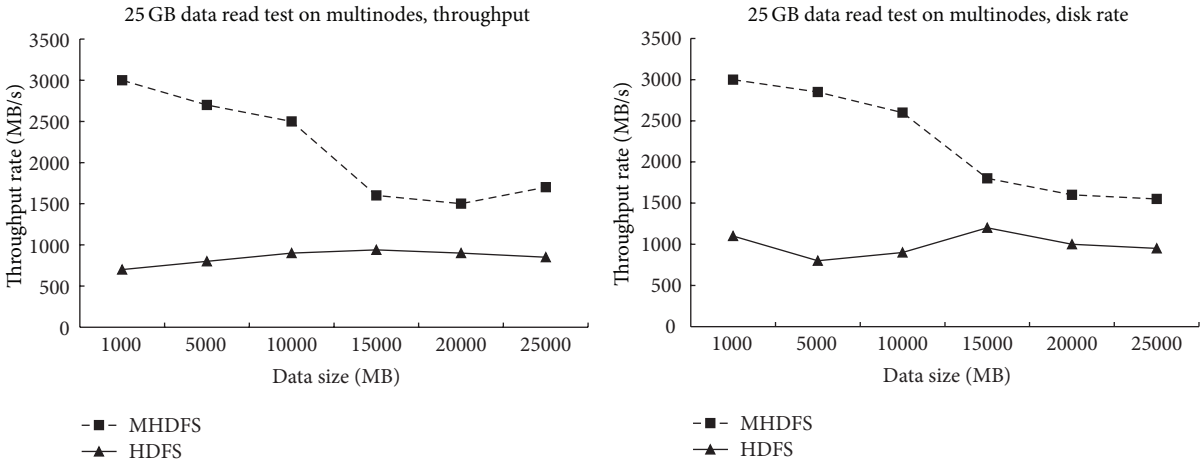


Figure 10: HDFS/MHDFS 25 GB data read test on multinode cluster.

The reading test result is undoubtedly consistent with other evaluations according to Figure 10, and, due to the factor of data locality, reading performance in multinode cluster declines faster than that in single-node situation. In detail, for single-node clusters, data is only read from local machine while, for multinode clusters, data shall occasionally be obtained from other machines via network.

5.4. A Real Case Benchmark. In this section, we benchmark our MHDFS framework compared with native HDFS based on the word-count case, that is, to evaluate the average performance when processing real-world applications. We evaluate the total executing time of the word-count cases as MapReduce jobs under different input sizes, and both the single-node and multinode clusters are examined (for the single-node case, we only use the name node as a stand-alone cluster). Figures 11 and 12 present the results of MHDFS and HDFS on various data sizes, respectively.

It is obvious that, on both single-node and multinode clusters, MHDFS achieves significant performance against traditional HDFS. This proves the improvement of our memory-based Hadoop framework that can accelerate the execution of MapReduce jobs with the advantage of storing the input data and intermediate results into the memory space. Generally, the MapReduce process of word count can

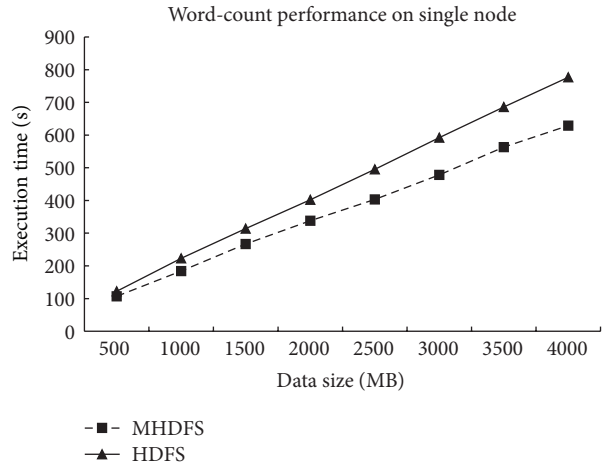


Figure 11: HDFS/MHDFS word-count test on single-node cluster.

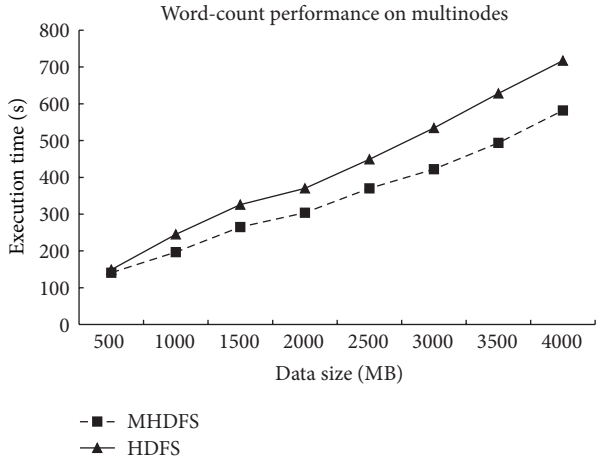


Figure 12: HDFS/MHDFS word-count test on multinode cluster.

generate double the sizes of its input; thus, for smaller size of input, MHDFS is quite close to HDFS, while, for larger size of input, MHDFS can notably benefit from the memory-based intermediate data storage.

The result on multinode cluster also indicates the overall better performance of MHDFS with the increasing of data size from 500 MB to 4 GB. Interestingly, due to the inferior capacities of data nodes, both MHDFS performance and HDFS performance are worse than that on single-node cluster from 500 MB to 1.5 GB. Since data processing is dispatched to various machines in a multinode cluster, the capability of machine is the major factor for smaller size of data, whereas, for larger size of data, it is the ability of parallel processing that counts, and thus HDFS and MHDFS achieve better performance than single-node situation, between which MHDFS costs less executing time owing to the employment of memory space.

6. Related Work

Despite its popularity and enormous applications, Hadoop has been the object of severe criticism, mainly due to its disk-based storage model [13]. The input and output data is required to be stored into the disk. Meanwhile, the inner mechanism for fault tolerance in Hadoop requires the intermediate results to be flushed into the hardware disk, which undoubtedly deteriorates the overall performance of processing large amount of data. In the literature, the existing improvement for Hadoop framework can be concluded as two categories: one aims at developing a wholly in-memory system that stores all the intermediate results into the memory, while the other focuses on employing a number of optimizations to make each usage of data more efficient [14, 15].

Spark [9, 10] and HANA [8] are typical examples belonging to the first category. They are totally in-memory systems that utilize the high throughput of memory media. Spark is a fast and general-purpose cluster computing system which provides high-level APIs and an optimized engine that supports general job executions. Resilient distributed datasets (RDD) are the key component for Spark to effectively manage the data transformation and storage in memory. HANA is the platform developed by SAP Inc. and is widely used in fast data processing by employing a specifically designed framework of memory-based distributed storing. The main shortcoming is that they are built upon another model completely different from native Hadoop which makes business migration inconvenient for companies. Meanwhile, the absence of disk utilization, to some extent, causes the systems' intensive dependence on limited memory spaces and lack of robust and fault tolerance. However, for those Hadoop-enhanced systems, these problems do not exist.

The second category, however, handles the problem from a different perspective. Hu et al. [15] analyzed the multiple-job parallelization problem and proposed the multiple-job optimization scheduler to improve the hardware utilization by paralleling different kinds of jobs. Condie et al. [16] added pipelined job interconnections to reduce the communication cost and lower the reading overhead from disk hardware. Since these researches mostly focus on designing probable scheduling strategies or intermediate result storing strategies, it is not necessary to enumerate them all, and although they can improve the performance to some extent, these works do not touch upon the idea of deploying Hadoop based on memory spaces.

Our research is distinct from the existing literatures mainly in the following aspects. First, we build MHDFS solely upon native Hadoop system without any modification to the crucial interfaces. This makes MHDFS compatible with currently existing systems and applications and makes migration much easier. Second, we design the strategy of memory data replacement to handle the situation when memory is fully occupied. Disk storages are still effectively utilized to improve the overall performance of our system.

7. Conclusion

As the basic foundation of Hadoop ecosystem, disk-dependent HDFS has been widely employed in various applications. Considering its shortcomings, mainly in low performance of the disk-based storage, we designed and implemented a memory-based Hadoop distributed file system named MHDFS. We proposed the framework of allocating reasonable memory space for our model and employed *Ramfs* tool to develop the system. Besides, the strategy of memory data replacement is studied for handling the insufficiency of memory space. Experiment results on different size of data indicate the effectiveness of our model. Our future study will focus on the further improvement of our proposed model and study the performance in more complicated situations.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grants nos. 61370207, 61572128,

and 61320106007, China National High Technology Research and Development Program (2013AA013503), SGCC Science and Technology Program "Research on the Key Technologies of the Distributed Data Management in the Dispatching and Control System Platform of Physical Distribution and Logical Integration," Collaborative Innovation Center of Wireless Communications Technology, Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu Provincial Key Laboratory of Network and Information Security (BM2003201), and Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grant no. 93K-9.

References

[1] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, OSDI, San Francisco, Calif, USA, 2004.

[2] D. Howe, M. Costanzo, P. Fey et al., "Big data: the future of biocuration," *Nature*, vol. 455, no. 7209, pp. 47–50, 2008.

[3] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Proceedings of the IEEE Grid Computing Environments Workshop (GCE '08)*, pp. 1–10, Austin, Tex, USA, November 2008.

[4] Apache Hadoop [EB/OL], http://hadoop.apache.org.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '07)*, pp. 59–72, Lisbon, Portugal, March 2007.

[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[7] J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[8] SAP, HANA[EB/OL], http://www.saphana.com/.

[9] SPARK, [EB/OL], http://www.sparkada.com/.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud '10)*, USENIX Association, 2010.

[11] B. Hindman, A. Konwinski, M. Zaharia et al., "Mesos: a platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, Boston, Mass, USA, April 2011.

[12] M. Zhao and R. J. Figueiredo, "Experimental study of virtual machine migration in support of reservation of cluster resources," in *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing (VTDC '07)*, ACM, November 2007.

[13] C. Doulkeridis and K. Nørvåg, "A survey of large-scale analytical query processing in MapReduce," *The VLDB Journal*, vol. 23, no. 3, pp. 355–380, 2014.

[14] A. Shinnar, D. Cunningham, V. Saraswat, and B. Herta, "M3R: increased performance for in-memory Hadoop jobs," *Proceedings of the VLDB Endowment* , vol. 5, no. 12, pp. 1736–1747, 2012.

[15] W. Hu, C. Tian, X. Liu et al., "Multiple-job optimization in mapreduce for heterogeneous workloads," in *Proceedings of the 6th International Conference on Semantics Knowledge and Grid (SKG '10)*, pp. 135–140, IEEE, Beijing, China, November 2010.

[16] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," in *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)*, vol. 10, p. 20, 2010.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 7241928, 13 pages
http://dx.doi.org/10.1155/2016/7241928

Research Article

Feedback-Based Resource Allocation in MapReduce-Based Systems

Bunjamin Memishi,[1] María S. Pérez,[1] and Gabriel Antoniu[2]

[1]*OEG, ETS de Ingenieros Informáticos, Universidad Politécnica de Madrid, Campus de Montegancedo, s/n Boadilla del Monte, 28660 Madrid, Spain*
[2]*Inria Rennes-Bretagne Atlantique Research Centre, Campus Universitaire de Beaulieu, Rennes, 35042 Brittany, France*

Correspondence should be addressed to Bunjamin Memishi; bmemishi@fi.upm.es

Received 14 January 2016; Accepted 28 March 2016

Academic Editor: Zhihui Du

Containers are considered an optimized fine-grain alternative to virtual machines in cloud-based systems. Some of the approaches which have adopted the use of containers are the MapReduce frameworks. This paper makes an analysis of the use of containers in MapReduce-based systems, concluding that the resource utilization of these systems in terms of containers is suboptimal. In order to solve this, the paper describes AdaptCont, a proposal for optimizing the containers allocation in MapReduce systems. AdaptCont is based on the foundations of feedback systems. Two different selection approaches, Dynamic AdaptCont and Pool AdaptCont, are defined. Whereas Dynamic AdaptCont calculates the exact amount of resources per each container, Pool AdaptCont chooses a predefined container from a pool of available configurations. AdaptCont is evaluated for a particular case, the application master container of Hadoop YARN. As we can see in the evaluation, AdaptCont behaves much better than the default resource allocation mechanism of Hadoop YARN.

1. Introduction

One of the most relevant features of cloud is virtualization. Many cloud infrastructures, such as Amazon EC2, offer virtual machines (VMs) to their clients with the aim of providing an isolated environment for running their processes. MapReduce systems [1] are also important cloud frameworks that can benefit from the power of virtualization. Nevertheless, VMs are extremely complex and heavyweight, since they are intended to emulate a complete computer system. This capability is not needed in MapReduce systems, since they only have to isolate the map and reduce processes, among other daemons. For this reason, containers, a much more lightweight virtualization abstraction, are more appropriate. Containers support the virtualization of a single application or process, and this is enough for MapReduce systems. Due to their nature, mainly by sharing a unique operating system kernel in a host, and being infrastructure independent, containers can start and terminate faster, which makes the container virtualization very efficient.

A container represents a simple unit of a box-like packed collection (or encapsulation) of resources, placed on a single node of a cluster. Whereas it shares many similarities with a VM, it also differs in some essential aspects. First, the container can represent a subset of a VM; conceptually, the VM could also be subset of a large container, but the practice suggests that it is better to avoid this scenario. The virtualization level is another crucial difference. VMs are designed to emulate virtual hardware through a full operating system and its proper additional add-ons, at the expense of more overhead. On the other hand, containers can easily use and share the host operating system, because they are envisioned to run a single application or a single process. Similarities between a container and VM are strongly linked in the manner of how they use resources. As in any VM, the main resources of a container are the main memory (RAM) and the computing processing unit (CPU).

The data storage and the data bandwidth are left in a second place.

Due to the less overhead of containers, a considerable number of cloud solutions, not only MapReduce-based clouds, are using currently these abstractions as resource allocation facility. Indeed, many experts are seeing containers as a natural replacement for VMs in order to allocate resources efficiently, although they are far from providing all the features needed for virtualizing operating systems or kernels. However, the coexistence between both abstractions, containers and VMs, is not only a feasible future but indeed now a reality.

According to our analysis made in Hadoop YARN [2], its containers allocation is not efficient. The current form of resource allocation at container level in Hadoop YARN makes it impossible to enforce a higher level of cloud elasticity. Elasticity can be defined as the degree to which a cloud infrastructure is capable of adapting its capacity to different workloads over time [3]. Usually, the number of containers allocated is bigger than needed, decreasing the performance of the system. However, occasionally, containers do not have sufficient resources for addressing the request requirements. This could lead to unreliable situations, jeopardizing the correct working of the applications. For the sake of simplicity, we only consider the main computing resources, the main memory (RAM), and the computing processing unit (CPU).

We present a novel approach for optimizing the resource allocation at the container level in MapReduce systems. This approach, called *AdaptCont*, is based on feedback systems [4], due to its dynamism and adaptation capabilities. When a user submits a request, this framework is able to choose the amount of resources needed, depending on several parameters, such as the real-time request input, the number of requests, the number of users, and the dynamic constraints of the system infrastructure, such as the set of resources available. The dynamic reaction behind the framework is achieved thanks to the real-time input provided from each user input and the dynamic constraints of the system infrastructure. We define two different selection approaches: Dynamic AdaptCont and Pool AdaptCont. Whereas Dynamic AdaptCont calculates the exact amount of resources per each container, Pool AdaptCont chooses a predefined container from a pool of available configurations.

In order to validate our approach, we use AdaptCont for a particular case study on a particular MapReduce system, the Hadoop YARN. We have chosen the application master of Hadoop YARN instead of the YARN workers, because of the importance of this daemon and because it involves the most complex use of containers. The application master container is required in every application. Additionally, the master orchestrates its proper job, but its reliability can jeopardize the work of the job workers. On the other hand, a particular worker usually does not have impact on the reliability of the overall job, although it may contribute to the delay of the completion time. The experiments show that our approach brings about substantial benefits compared to the default mechanism of YARN, in terms of use of RAM and CPU. Our evaluation shows improvements in the use of these resources, which range from 15% to 75%.

In summary, this paper has the following main contributions:

(1) Definition of a general-purpose framework called AdaptCont, for the resource allocation at the container level in MapReduce systems.

(2) Instantiation of AdaptCont for a particular case study on Hadoop YARN, that is, the application master container.

(3) Evaluation of AdaptCont and comparison with the default behavior of Hadoop YARN.

The rest of the paper is organized as follows. In Section 2, we introduce AdaptCont as a general framework based on feedback systems for allocating container resources. We introduce a case study of the framework in Section 3. We evaluate AdaptCont in Section 4. In Section 5, we discuss the related work. Finally, we summarize the main contributions and outline the future work in Section 6.

2. AdaptCont Framework

According to [4], feedback systems refer to two or more dynamical systems, which are interconnected in such a way that each system affects the behavior of others. Feedback systems may be open or closed. Assuming a feedback system F, composed of two systems A and B, F is closed if their components form a cycle, with the output of system A being the input of system B and the output of system B the input of system A. On the contrary, F is open when the interconnection between systems B and A is broken.

Feedback systems are based on a basic principle: correcting actions should always be performed on the difference between the desired and the actual performance. Feedback allows us to (i) provide robustness to the systems, (ii) modify the dynamics of a system by means of these correcting actions, and (iii) provide a higher level of automation. When a feedback system is not properly designed, a well known drawback is the possibility of instability.

An example of a dynamic system that can benefit from the feedback theory nowadays is a production cloud [5]. In this scenario, users, applications, and infrastructure are clearly interconnected and the behaviors of any of these systems influence each other. Our approach, AdaptCont, is a feedback system, whose main goal is to optimize the resource allocation at the container level in clouds and specifically in MapReduce-based systems.

Before designing the feedback system, it is necessary to define the features of a cloud:

(i) A cloud has a limited set of nodes $n_1, n_2, \ldots, n_m$.

(ii) Each node n_i has a limited set of containers $c_i1, c_i2, \ldots, c_il$.

(iii) The system can receive a limited set of job requests $j_1, j_2, \ldots, j_r$.

(iv) Every job request has its workload input. These jobs are part of applications.

(v) The same workload can be used as an input for different applications.

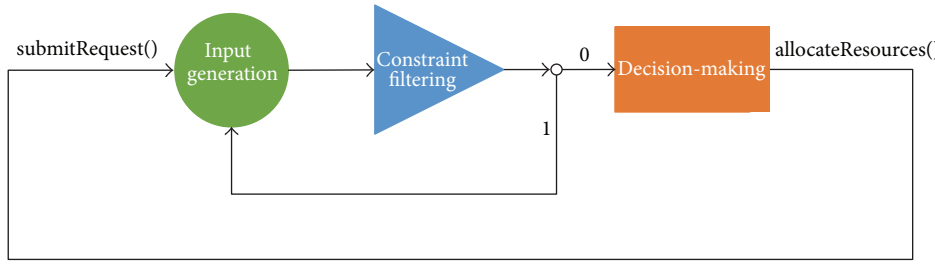


Figure 1: A generalized framework for self-adaptive containers, based on the feedback theory.

(vi) Applications could divide a large workload into small input partitions called *splits*, each split being a workload of a particular container.

(vii) Depending on the cluster size and scheduler limitations, simultaneous containers could run in single or multiple sequential groups called *waves*.

(viii) By default, all the containers should finish before the application submits the final output to the user.

(ix) Applications may introduce different job completion time, though under the same user, input, and allocated resources.

In a dynamic cloud, these parameters may change in real time. Detecting these changes is strongly dependent on the monitoring system, which should be particularly focused on the infrastructure [6].

At a generic level, we can follow a feedback-based approach based on three stages: input generation, constraint filtering, and decision-making. The general pattern is shown in Figure 1. This approach is closed. In real time, the input generation module could receive several constraints in sequence. After generating the initial parameters (by taking into account the initial constraints), an additional follow-up constraint may require another parameters calculation before being sent to the decision-making module. Consequently, the number of runs of the input generation module is proportional to the modifications (constraints) identified from the system.

2.1. Input Generation. The input generation module of AdaptCont collects or generates the required parameters for making decisions about efficient resource allocation. These parameters are as follows:

(i) The input workload size.

(ii) The input split size enforced by the application.

(iii) The total number of available containers per each user.

(iv) The wave size in which these containers may be run.

(v) The constraints introduced by users.

Some of these parameters are collected directly from the application. For instance, the input workload size comes in every job request. Other parameters are more complex to be generated. For instance, the number of waves w depends on the number of input splits n_s and the number of available containers per user n_c, being calculated as $w = n_s/n_c$.

2.2. Constraint Filtering. This stage is needed because clouds have a limited number of costly resources. Constraints may be imposed by the infrastructure, application, and/or users.

Infrastructure constraints are those constraints related to the limitation of the cloud provider, since not always the number of resources is enough for fulfilling the resource requests of all the applications and users.

Some constraints are enforced by applications. For instance, some applications require a certain type of sequential container. This is the case of MapReduce systems, where, by default, containers of the first phase (map) need to finish before the containers of the second phase (reduce) start [7, 8].

Finally, other constraints are defined by users. For instance, some users have a limited capability for buying resources.

2.3. Decision-Making. Based on the parameters coming from the previous modules, the decision-making module outputs the final resource allocation. In particular, this module decides the minimum recommended container memory c_{RAM} and CPU power c_{CPU} per every container. This decision depends on the particular problem addressed by these containers.

Once this module has decided these values for a specific application of a user, the rest of the process is automatic, since all the containers of an application are equal. This process has to be called for different applications or different users.

2.4. Predefined Containers. A possible improvement of AdaptCont is enabling the use of predefined containers with different configurations (e.g., small, medium, and large). This means that a cloud has a pool of static containers that can be used for different user request. In this way, it will not be necessary to trigger a new container, but a predefined one ready to be used. This reduces the overhead of the resource allocation process during the job submission. This feature should be part of the decision-making module.

How can the framework define this pool of containers? First, it should be able to identify the typical user requests in the system. These requests may be evaluated from (i) previous (stored) monitoring values or from (ii) other monitoring variables measured at the same time, according to [9].

What happens if the container does not have the exact configuration we need? In this case, the decision-making module establishes a threshold. If the difference between the required and existing configurations is below this threshold,

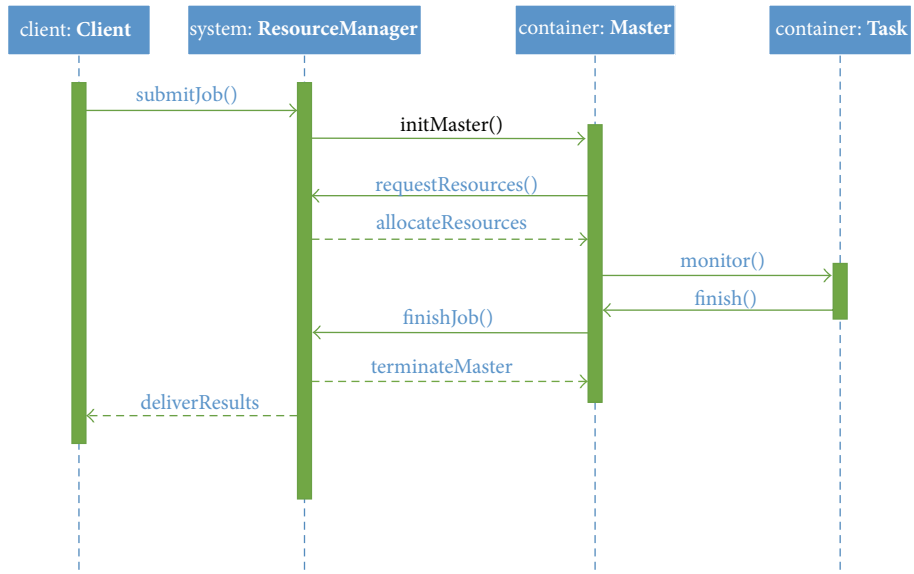


Figure 2: Job flow messages in Hadoop YARN: a sequence diagram.

the system uses the already existing container. Otherwise, the system triggers a new container.

3. AdaptCont Applied to YARN

We have chosen as a case of study the analysis of a relevant type of a container in a specific kind of cloud systems, that is, MapReduce-based clouds. Namely, the chosen container is the application master in the next-generation MapReduce system called YARN [2].

3.1. Background. YARN constitutes the new version of Apache Hadoop. This new implementation was built with the aim of solving some of the problems shown by the old Hadoop version. Basically, YARN is a resource management platform that, unlike the former Hadoop release, provides greater scalability and higher efficiency and enables different frameworks to efficiently share a cluster. YARN offers, among others, MapReduce capabilities.

The basic idea behind YARN is the separation between the two main operations of the classic Hadoop master, resource management and job scheduling/monitoring, into separate entities or daemons. The resource manager consists of two main components: the *scheduler* and the *application manager*. While the scheduler's duty is resource allocation, the application manager accepts job submissions and initiates the first job container for the application master. After this, the job is managed by the application master, which starts negotiating resources with the resource manager and collaborates with the node managers to run and monitor its tasks. Finally, it informs the resource manager that has been completed and releases its container. The resource manager delivers the results to the client. A simple sequence of these steps is given in Figure 2.

For each job submission, the application master configuration is static and does not change for different scenarios. According to the state-of-the-art literature [10–14], most large-scale MapReduce clusters run small jobs. As we will show in Section 4, even the smallest resource configuration of the application master exceeds the requirements of these workloads. This implies a waste of resources, which could be alleviated if the configuration is adapted to the workload size and the infrastructure resources. Moreover, some big workloads could fail if the container size is not enough for managing them. At large-scale level, this would have a higher impact. Therefore, our goal is to choose an appropriate container for the application master.

3.2. Design. In order to optimize containers for the application master, we will follow the same pattern of the general framework, that is, AdaptCont.

The input generation module divides the workload input size into splits. The YARN scheduler provides containers to users, according to the number of available containers of the infrastructure each instant of time. As we mentioned above, the input generation module calculates the number of waves from the number of input splits and the number of available containers per user. Figure 3 shows how the application master manages these waves.

Many constraints can be raised from the scheduler. An example of this is the phase priority. It is well known that the map phase input is by default bigger than or equal to the reduce phase input [15]. This is one of the reasons why the number of mappers is higher than the number of reducers. Due to this, as a reasonable constraint, the constraint filtering module prioritizes the number of mappers with regard to the number of reducers.

Decision-making module considers mainly two parameters, total workload and wave sizes. Contrary to what it may seem at first sight, the type of application does not affect the resource allocation decision of our use case. Some applications could have more memory, CPU, or I/O

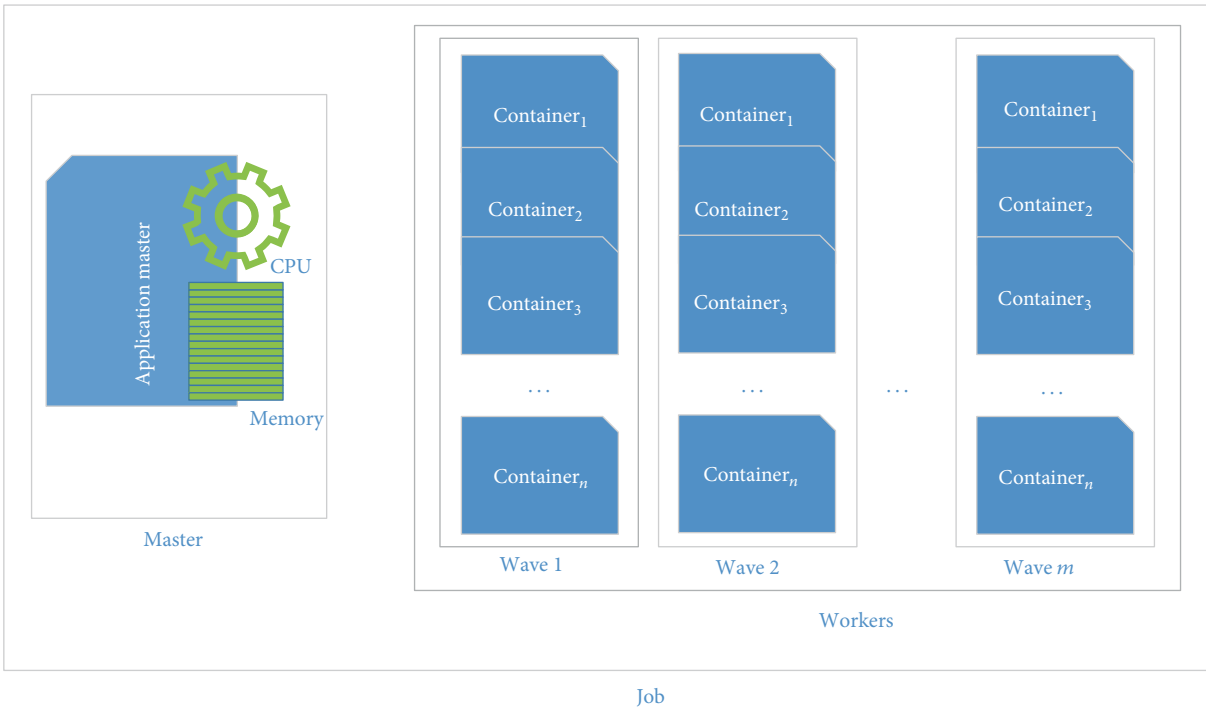


Figure 3: Workers containers monitored in waves by the application master container.

requirements, influencing the number and types of needed containers. However, this would only determine the size of the worker containers, and, in this case study, our scope is focused only on the master containers, which contribute largely to the reliability of the application executions.

Decision-making module uses two parameters: Ω and Ψ. The first parameter represents the minimum recommended memory size for an application master container that manages one unit wave, w_{unit}. Our goal is to calculate c_{RAM} from the value of Ω, with c_{RAM} being the recommended memory size for the application master. In the same way, we aim to calculate c_{CPU} as the recommended CPU power for the application master, from Ψ, which is the minimum recommended CPU power for an application master that manages w_{unit}.

To calculate the memory, if the actual wave w is bigger than what could be handled by Ω, that is, bigger than w_{unit}, then we declare a variable λ that measures this wave magnitude: $\lambda = w/w_{\text{unit}}$. Now, it is easy to find c_{RAM}:

$$c_{\text{RAM}} = \lambda * \Omega + Stdev, \quad Stdev \in \left[0; \frac{\Omega}{2}\right]. \tag{1}$$

Regarding the CPU power, the formula for c_{CPU} is

$$c_{\text{CPU}} = \lambda * \Psi + Stdev, \quad Stdev \in \left[0; \frac{\Psi}{2}\right]. \tag{2}$$

Figure 4 represents the AdaptCont modules, which are executed in the context of different YARN daemons. Whereas the input generation and the decision-making modules are part of the application manager, the constraint filtering module is part of the scheduler. The combination of both daemons forms the resource manager. The resource manager has a complete knowledge about each user through the application manager and the available resources through the scheduler daemon. When the application manager receives a user request, the resource manager is informed about the workload input. The scheduler informs the application manager of every important modification regarding the monitored cluster. According to this, the application manager reacts upon the user request, by optimizing the container for its application master.

4. Experimental Evaluation

We have performed a set of experiments to validate our approach and compare it with YARN. These experiments have been made by means of simulations. In order to make this evaluation, we have followed the methodology of Section 4.1. Results of the evaluation are described in Section 4.2. Finally, the discussion about these results is shown in Section 4.3.

4.1. Methodology. To evaluate AdaptCont, we have considered three different schedulers and three different application master configurations, as is shown in Table 1. Below we give details for all of them.

Scheduler. We have taken into account three important schedulers, already implemented in YARN:

(i) *FIFO Scheduler*. This was the first scheduling algorithm that was implemented for MapReduce. It works on the principle that the master has a queue of jobs, and it simply pulls the oldest job first.

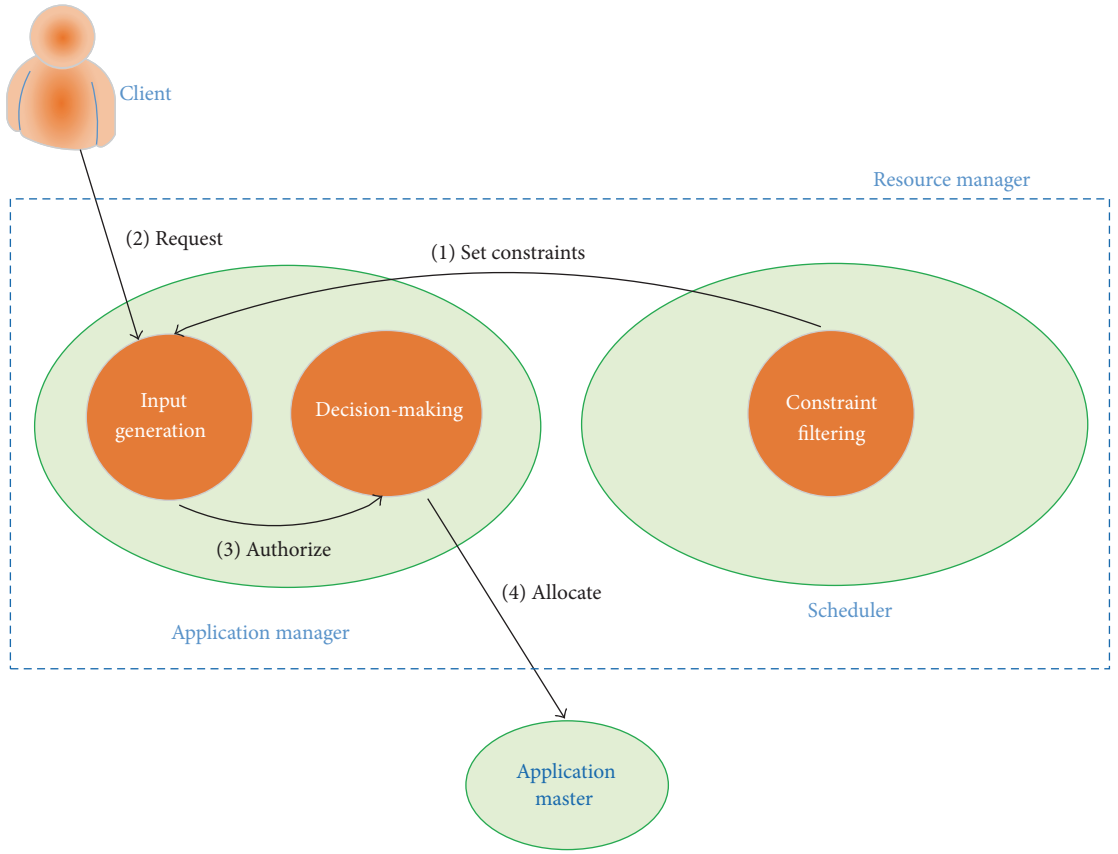


Figure 4: AdaptCont model applied to the Hadoop YARN application master.

Table 1: Methodology description, taking into account different schedulers and masters.

Scheduler	Master		
	YARN	Dynamic	Pool
FIFO	FIFO-YARN	FIFO-Dynamic	FIFO-Pool
Fair	Fair-YARN	Fair-Dynamic	Fair-Pool
Capacity	Capacity-YARN	Capacity-Dynamic	Capacity-Pool

FIFO: FIFO scheduler. Fair: Fair scheduler. Capacity: Capacity scheduler. YARN: YARN master. Dynamic: Dynamic master. Pool: Predefined containers-based master.

(ii) *Fair Scheduler*. It assigns the same amount of resources (containers) to all the workloads, so that on average every job gets an equal share of containers during its lifetime.

(iii) *Capacity Scheduler*. It gives different amount of resources (containers) to different workloads. The bigger the workload is, the more the resources are allocated to it.

Master. To compare YARN with AdaptCont, we use the following application master configurations:

(i) *YARN Application Master (YARN)*. This is the default implementation of the application master in YARN.

(ii) *Dynamic Master (Dynamic AdaptCont)*. This master container is adjusted in accordance with AdaptCont. Namely, it calculates the memory and CPU, according to the decision-making module and only after this does it initiate the master.

(iii) *Predefined Containers-Based Master (Pool AdaptCont)*. As defined in Section 2.4, the resource manager has a pool of master containers, which can be allocated depending on the workload size. This is an optional optimization of AdaptCont.

Workload. According to the job arrival time, we consider two additional sets of experiments:

(i) *Set-All*. In this scenario, all the jobs are already in the queue of the scheduler. We are going to combine this scenario with all the values of Table 1, since it is important to evaluate the approach under pressure, that is, when the load reaches high values.

(ii) *Set-Random*. This is a more realistic scenario, where jobs arrive at random times. Again, this scenario is evaluated in combination with all the values of Table 1, in order to simulate the behavior of a common MapReduce cluster.

An important parameter to take into account is the workload size. We introduce two additional scenarios:

(i) *Workload-Mixed*. In this case, the workload size will be variable, ranging from 500 MB to 105 GB, taking (1) 500 MB, (2) 3.5 GB, (3) 7 GB, (4) 15 GB, (5) 30 GB, (6) 45 GB, (7) 60 GB, (8) 75 GB, (9) 90 GB, and (10) 105 GB as workload size inputs. We have used these boundaries, because of the average workload sizes of important production clusters. For instance, around 90% of workload inputs in Facebook [12] are below 100 GB.

(ii) *Workload-Same*. In this case, every input (10 workloads) is the same: 10 GB. We have used this value, since, on average, the input workloads at Yahoo and Microsoft [12] are under 14 GB.

Therefore, we evaluate AdaptCont with the values of Table 1 and the 4 combinations from previous scenarios: *Set All-Workload Mix*, *Set All-Workload Same*, *Set Random-Workload Mix*, and *Set Random-Workload Same*.

Constraints. In MapReduce, the application master has to manage both map and reduce workers. The map phase input is always bigger than or equal to the reduce phase input [15]. This is one of the reasons why the number of mappers is bigger than the number of reducers. On the other hand, both phases are run sequentially. Thus, we can assume as constraint that the master container resources depend on the number of mappers and not on the number of reducers.

In order to simulate a realistic scenario, we have introduced in our experiments a partition failure that will impact around 10% of the cluster size. We assume that this failure appears in the fifth iteration (wave). This constraint forces AdaptCont to react in real time and adapt itself to a new execution environment, having to make decisions about future resource allocations.

Setup. In our experiments, 250 containers are used for worker tasks (mappers and reducers). This number of containers is sufficient to evaluate the approach, considering 25 containers per workload. We consider that every map and reduce container is the same and can execute a particular portion (split) of the workload. Each task runs on a container that has 1024 MB RAM and 1 virtual core. According to [16–18], a physical CPU core is capable of giving optimal performance of the container, if it simultaneously processes 2 containers at most. Therefore, we take 1 CPU core as equivalent to 2 virtual cores.

Our goal is to evaluate the resource utilization of the application masters, in terms of CPU and RAM. To get this, we consider an isolated set of resources oriented only to application masters. In this way, it will be easier to measure the impact of AdaptCont on saving resources.

4.2. Results. In this section, we compare the CPU and memory efficiency of YARN versus Dynamic AdaptCont and Pool AdaptCont. Before that, we analyze the wave behavior of the 10 workloads.

Wave Behavior. Figure 5 represents the resource allocation (maximum number of containers or wave sizes) for the combination we have mentioned before: *Set All-Workload Mix*, *Set All-Workload Same*, *Set Random-Workload Mix*, and *Set Random-Workload Same*.

Figure 5(a) shows different workload sizes with the same arrival time (already in the scheduler queue). The experiments demonstrate that a maximum wave is dependent on the workload size and the scheduler. Regarding the FIFO scheduler, since the queue order is formed by the smallest workload first, for these small workloads, the maximum wave is represented by the needed containers. For instance, the first workload needs only 8 containers. This number of containers is calculated dividing the workload size by the split size (64 MB). These 8 containers are provided by the infrastructure, and this is the case of the second workload (56 containers) and the third workload (112 containers). For the fourth workload, the infrastructure is not capable of providing the needed containers, which only has 74 containers in the first wave, that is, $250 - (8 + 56 + 112)$. The fourth workload needs 240 containers in total. Thus, the remaining containers ($240 - 74 = 166$) will be provided in the next wave.

In the second wave, since the first three workloads have finished, the scheduler will provide 166 containers to the fourth workload and the rest ($250 - 166 = 84$) to the fifth workload. This process is repeated until all the workloads are given the necessary containers and every job has terminated. As we can notice, the maximum wave for the latest workloads reaches higher amount of allocated containers, since the workload is bigger, and in most of the cases the scheduler is busy with a unique job. Although initially the infrastructure has 250 containers, from the fifth wave, there is a slight decrease (225), due to the partition failure (10% of the resources). This only affects the workloads not having finished before this wave (in this case, the fifth).

The main drawback of the FIFO scheduler is that it may delay the completion time of the smallest jobs, especially if they arrive late to the queue. In general, this scheduler is not fair in the resource allocation and depends exclusively on the arrival time.

Regarding the fair scheduler, this scheduler allocates the same number of containers to all the workloads and consequently to all the users, that is, $250/10 = 25$. The partition failure forces the fair scheduler to decrease the number of containers to 22 (225/10) from the fifth wave.

With regard to the capacity scheduler, this scheduler takes advantage of available resources once some jobs have finished. At the beginning, it behaves like the fair scheduler. However, when some small jobs have terminated, the available resources can be reallocated to the rest of the workloads. This is the reason why the biggest workloads in the queue get a higher number of containers. As in the previous case, the partition failure also implies a slight decrease in the number of containers from the fifth wave.

Figure 5(b) represents the same mixed workloads but when they arrive randomly to the scheduler queue. Clearly, the main differences are noted in the FIFO scheduler, because the arrival time of the workloads is different and now one of the biggest workloads (9) appears in first place.

The other subplots of Figure 5 show the experimental results of the same workloads with an input of 10 GB. This

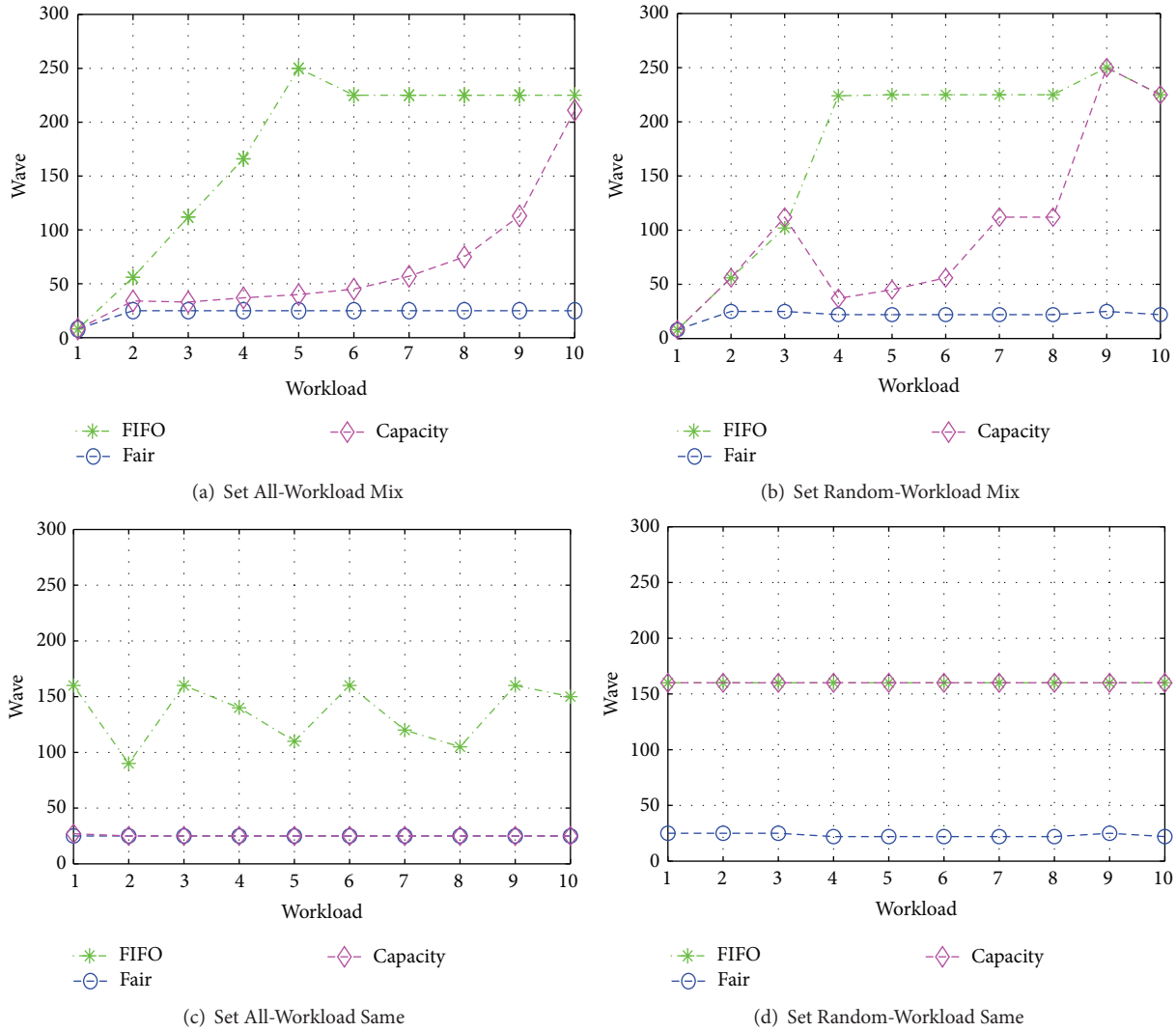


Figure 5: Wave behavior: wave size according to the scheduler and the workload type.

input requires a static number of containers (in this case, 160 containers).

In Figure 5(c), all the jobs have arrived to the queue. In this scenario, the FIFO allocation oscillates between the maximum wave of 160 containers and the smallest wave of 90 containers (250 − 160). This oscillation is caused by the allocation of resources to the previous workload, which does not leave enough resources for the next one, and then the cycle is repeated again.

In this case, the fair and capacity schedulers have the same behavior, since all the workloads are equal.

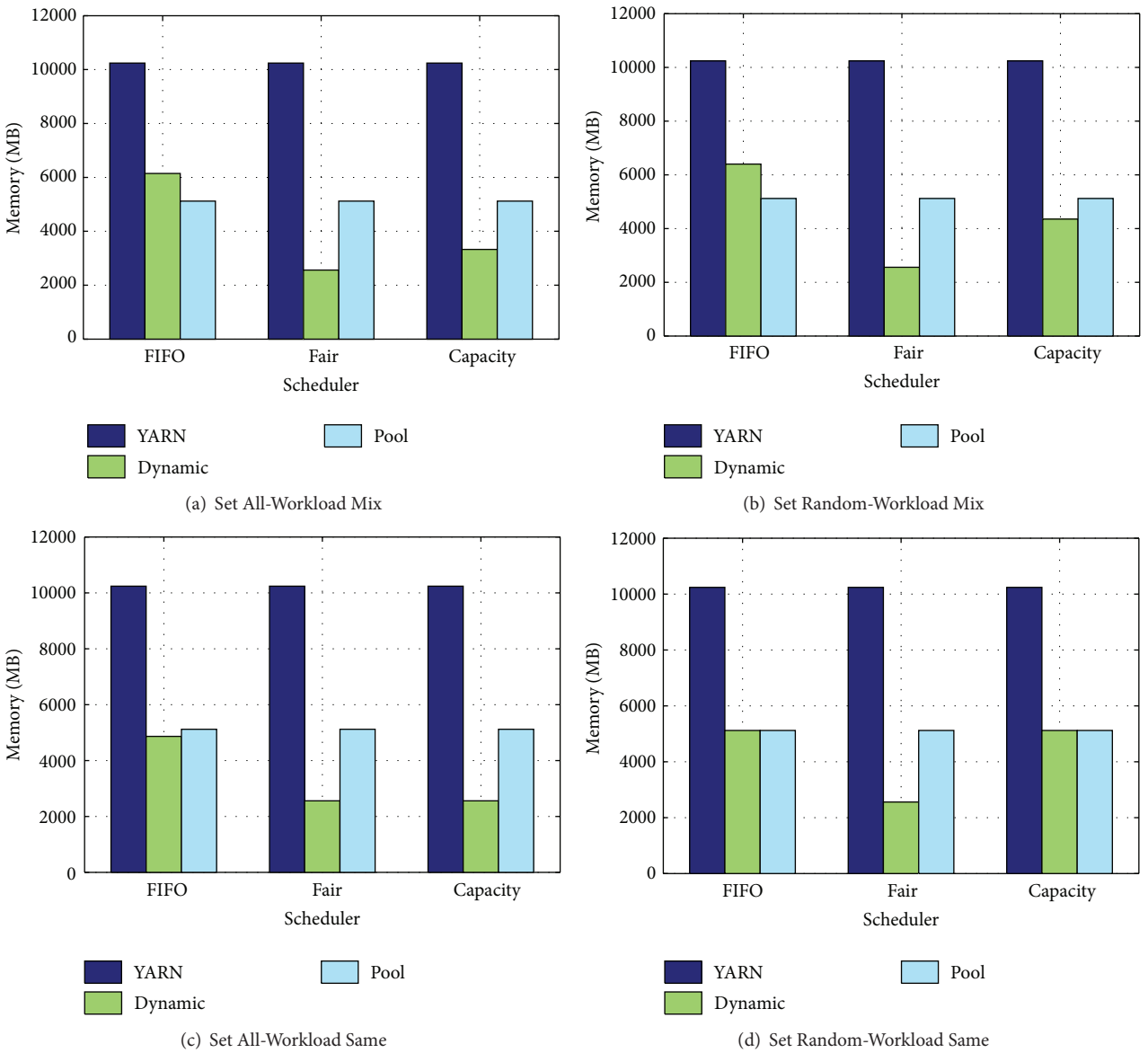
Figure 5(d) shows the number of containers for the same workload with random arrival. The difference of this scenario versus the scenario shown in Figure 5(c) is twofold:

(1) The arrival of these jobs is consecutive. In every wave, a job arrives. Due to this, the FIFO scheduler is forced to wait after each round for a new workload, even though at every round there are available resources (250 − 160 = 90), not allocated to any job. Thus, the FIFO scheduler always allocates 160 containers in every wave.

(2) Whereas, in the previous scenario, the fair and capacity schedulers behave the same, in this case, the capacity scheduler acts similarly to the FIFO scheduler. This is because the capacity scheduler adapts its decisions to the number of available resources, which is enough in every moment for addressing the requirements of the jobs (160 containers). Thus, the capacity scheduler achieves a better completion time, compared to the fair scheduler.

According to this analysis, we can conclude that the wave behavior and size are decisive in the application master configuration.

Memory Usage. Figure 6 shows for the 4 scenarios the total memory used by the three approaches: YARN, Dynamic AdaptCont, and Pool AdaptCont.



(a) Set All-Workload Mix

(b) Set Random-Workload Mix

(c) Set All-Workload Same

(d) Set Random-Workload Same

Figure 6: Memory usage and master type versus scheduler.

In the case of YARN, we have deployed the default configuration, choosing the minimum memory allocation for the application master (1024 MB).

The Dynamic AdaptCont-based application master memory is dependent on the waves size. If the wave size is under 100, the decision-making module allocates a minimum recommended memory of 256 MB. For each increase of 100 in the wave size, the memory is doubled. The reasons behind this are as follows:

(1) A normal Hadoop task does not need more than 200 MB [12], and this is even clearer in the case of the application master.

(2) As most of the jobs are small [12–14], consequently, the maximum number of mappers is also small and, therefore, the application master requires less memory.

(3) The minimum recommended memory by Hortonworks [17] is 256 MB.

The Pool AdaptCont-based application master works in a different way, constituting an alternative between the YARN master and the Dynamic master. This application master has three default configurations: *small*, *medium*, and *big*. The small master has 512 MB of memory, for all small jobs that need a maximum of 250 containers. The medium master has 1024 MB, as it is the default minimum YARN setting. In order to deal with big waves, the big configuration has 2048 MB.

As we can see in Figure 6, YARN is outperformed by both AdaptCont approaches. YARN always consumes 10 GB, not depending on the different use cases. For instance, in Figure 6(a), Dynamic AdaptCont has memory usage of 6144 MB versus 10 GB in YARN, achieving 40% memory

improvement. In this case, Pool AdaptCont only uses 5120 MB, that is, 50% improvement compared to YARN. This difference between Dynamic AdaptCont and Pool AdaptCont for the FIFO scheduler is due to the way of providing memory in both approaches. If the workload needs 250 containers, Dynamic AdaptCont provides $256\lceil(250/100)\rceil$ MB, that is, $256 * 3 = 768$ MB. In the same scenario, Pool AdaptCont provides 512 MB, corresponding to the small size configuration.

In general, Dynamic AdaptCont is the best approach in terms of memory usage, except in the case of the FIFO scheduler, where the performance is close to and slightly worse than the performance of Pool AdaptCont. In the case of fair and capacity schedulers, Dynamic AdaptCont is the best alternative, achieving on average 75% and 67.5% improvement compared to YARN, versus 50% improvement provided by Pool AdaptCont.

CPU Usage. The CPU usage is another relevant parameter to take into account. In order to measure it, we have correlated memory and CPU, considering that we need higher CPU power to process a larger amount of data, stored in memory.

In YARN, you can assign a value ranging from 1 up to 32 of virtual cores for the application master. This is also the possible interval allocation for every other container. According to [16], 32 is the maximum value. In our experiments, we use the minimum value for the YARN master (1 virtual core for its container) per 1024 MB.

For the Dynamic AdaptCont, the decision-making module increases the number of virtual cores after two successive increments of 256 MB of memory. This decision is based on the abovementioned methodology, which states that a physical CPU core is capable of giving optimal performance of the container, if it simultaneously processes 2 containers at most [16–18]. To be conservative, we address the smallest container, that is, a container of 256 MB. For instance, if the memory usage is 768 MB, the chosen number of virtual cores is 2.

The same strategy is valid for the Pool AdaptCont, assuming 1 virtual core for small containers, 2 virtual cores for medium containers, and 3 virtual cores for large containers.

Due to this policy, the CPU does not change so abruptly as the memory for Dynamic and Pool AdaptCont. Thus, as is shown in Figure 7, both approaches behave similarly, except in the case of FIFO with Workload Mix. This was previously justified in the memory usage evaluation. As the CPU is proportional to the memory usage, the behavior of Dynamic AdaptCont with FIFO for Workload Mix is again repeated in the case of CPU.

In most of the cases, the improvement of both Dynamic and Pool AdaptCont against YARN reaches 50%.

4.3. Discussion. In this section, we discuss what combination of approaches and schedulers can be beneficial in common scenarios.

As a result of the experiments, we can conclude that YARN used by default is not appropriate for optimizing the use of MapReduce-based clouds, due to the waste of resources.

In the presence of heavy and known advanced workloads (this is the usual case of scientific workloads), according to our results, the best recommended strategy is to use Dynamic AdaptCont combined with FIFO scheduler.

However, if we have limited resources per user, a better choice could be Dynamic AdaptCont combined with fair scheduler. This scheduler allocates a small set of resources to every workload, improving the overall performance.

In a scenario where we have a mixture of large and small workloads, the choice should be Dynamic AdaptCont combined with capacity scheduler. This is due to the adaptability of this scheduler with regard to the input workload and available resources.

Finally, as shown in the experiments, if our focus is on CPU and not on memory, we can decide to use Pool AdaptCont (combined with any schedulers) instead of the dynamic approach.

5. Related Work

As far as we know, this paper is the first contribution that proposes a MapReduce optimization through container management. In particular, linked to our use case, it is the first contribution that aims to create reliable masters, by means of the allocation of sufficient resources to their containers.

There are many contributions on MapReduce whose goal is optimizing the framework from different viewpoints. An automatic optimization of the MapReduce programs has been proposed in [19]. In this work, authors provide out-of-the-box performance for MapReduce programs that need to be run using as input large datasets. In [20], an optimization system called Manimal was introduced, which analyzes MapReduce programs by applying appropriate data-aware optimizations. The benefit of this *best-effort* system is that it speeds up these programs in an autonomic way, without human intervention. In [21], a new classifications algorithm is introduced with the aim of improving the data locality of mappers and the task execution time. All these contributions differ from our contribution since they are only software-oriented optimizations for the MapReduce pipeline, and they do not take into account the resource allocation or the CPU and memory efficiency.

FlexSlot [22] is an approach that resizes map slots and changes the number of slots of Hadoop in order to accelerate the job execution. With the same aim, DynamicMR [23] tries to relax the slot allocation constraint between mappers and reducers. Unlike our approach, FlexSlot is only focused on the map stage and both FlexSlot and DynamicMR do not consider the containers as resource allocation facility.

In [24], authors introduce MRONLINE, which is able to configure relevant parameters of MapReduce online, by collecting previous statistics and predicting the task configuration in fine-grain level. Unlike MRONLINE, AdaptCont uses a feedback-control approach that also enables its application to single points of failure.

Cura [25] automatically creates an optimal cluster configuration for MapReduce jobs, by means of the framework profiling, reaching global resource optimization. In addition, Cura introduces a secure instant VM allocation to reduce

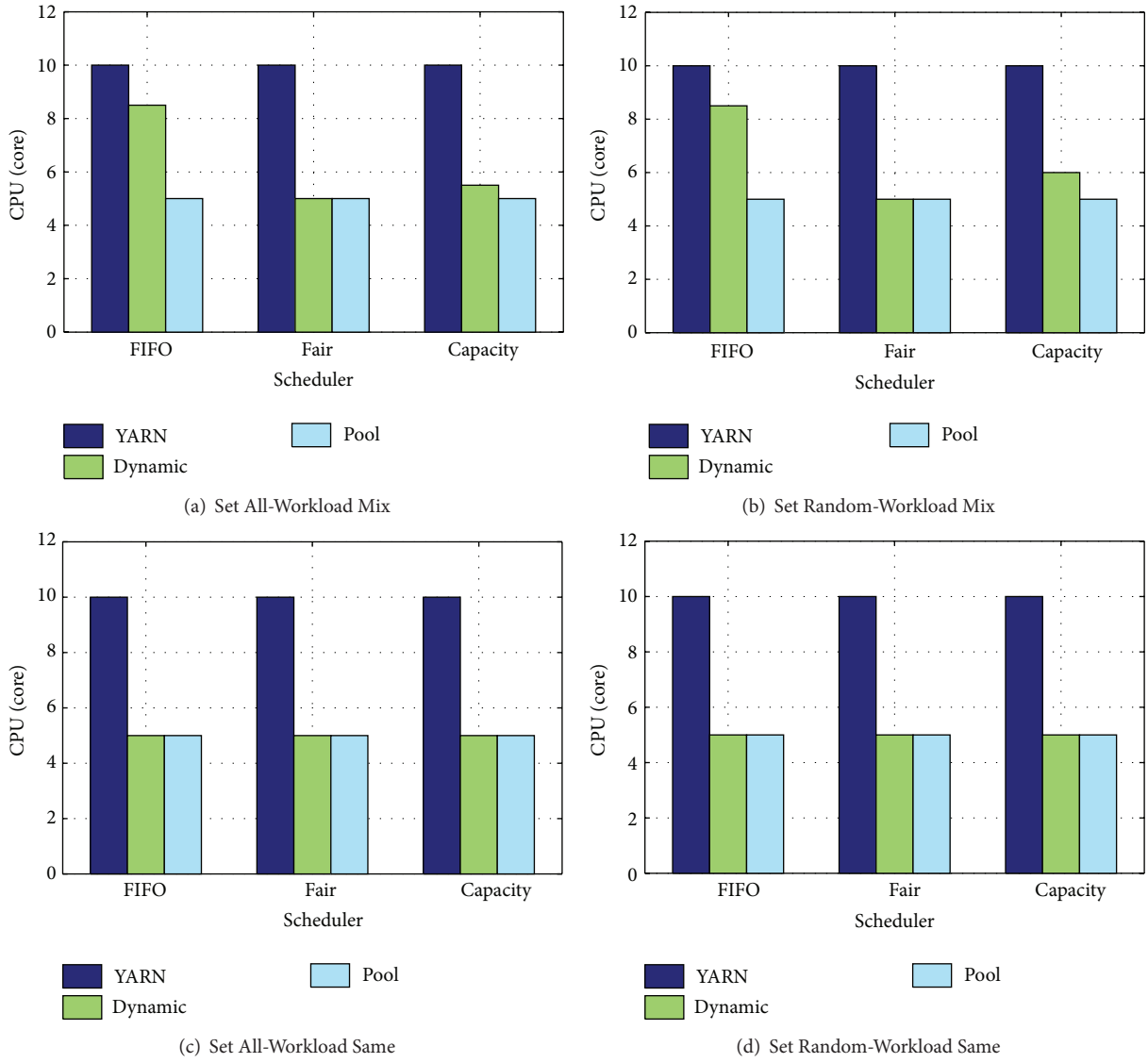


Figure 7: CPU usage and master type versus scheduler.

the response time for the short jobs. Finally, it applies other resource management techniques such as cost-aware resource provisioning, VM-aware scheduling, and online VM reconfiguration. Overall, these techniques lead to the enhancement of the response time and reduce the resource cost. This proposal differs from our work, because it is mostly concentrated in particular workloads excluding others. Furthermore, it is focused on VMs management and not on containers, as AdaptCont.

Other proposals aim to improve the reliability of the MapReduce framework, depending on the executional environment. The work proposed in [26] is a wider review that includes byzantine failures in Hadoop. The main properties upon which the UpRight library is based are safety and eventual liveliness. The contribution of this paper is to establish byzantine fault tolerance as a viable alternative to crash fault tolerance for at least some cluster services rather than any individual technique.

The work presented in [27] represents a byzantine fault-tolerant (BFT) MapReduce runtime system that tolerates faults that corrupt the results of computation of tasks, such as the cases of DRAM and CPU errors/faults. The BFT MapReduce follows the approach of executing each task more than once, but in particular circumstances. This implementation uses several mechanisms to minimize both the number of copies of tasks executed and the time needed to execute them. This approach has been adapted to multicloud environments in [28].

In [29], authors propose another solution for intentional failures called Accountable MapReduce. This proposal forces each machine in the cluster to be responsible for its behavior, by means of setting a group of auditors that perform an accountability test that checks the live nodes. This is done in real time, with the aim of detecting the malicious nodes.

In order to improve master reliability, [30] proposes to use a clone master. All the worker nodes should report their

activity to this clone master. For unstable environments, some other works [31–33] introduce dedicated nodes for the main daemons, including the master daemon.

Unlike our approach, these contributions related to reliability do not deal with the resource utilization.

6. Conclusions

The classic Apache Hadoop (MapReduce 1.0) has evolved for a long time by means of the release of several versions. However, the scalability limitations of Hadoop have only been solved partially with Hadoop YARN (MapReduce 2.0). Nevertheless, YARN does not provide an optimum solution to resource allocation, specifically at container level, causing both performance degradation and unreliable scenarios.

This paper proposes AdaptCont, a novel optimization framework for resource allocation at the container level, based on feedback systems. This approach can use two different selection algorithms, Dynamic AdaptCont and Pool AdaptCont. On the one hand, Dynamic AdaptCont figures out the exact amount of resources per each container. On the other hand, Pool AdaptCont chooses a predefined container from a pool of available configurations. The experimental evaluation demonstrates that AdaptCont outperforms the default resource allocation mechanism of YARN in terms of RAM and CPU usage, by a range of improvement from 40% to 75% for memory usage and from 15% to 50% for CPU utilization.

As far as we know, this is the first approach to improve the resource utilization at container level in MapReduce systems. In particular, we have optimized the performance of the YARN application master. As future work, we will explore the adaptation of AdaptCont for other containers of MapReduce worker tasks and deploy AdaptCont on real distributed infrastructures. We also expect to explore AdaptCont for VMs, in particular for allocating raw VMs to different user requests. We believe that fine-tuning a VM can be optimized, driven by requirements coming from an intersection between performance, reliability, and energy efficiency.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The research leading to these results has received funding from the H2020 project Reference no. 642963 in the call H2020-MSCA-ITN-2014.

References

[1] J. Dean, S. Ghemawat, and Google Inc, "MapReduce: simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation (OSDI '04)*, USENIX Association, San Francisco, Calif, USA, December 2004.

[2] V. K. Vavilapalli, A. C. Murthy, C. Douglas et al., "Apache hadoop YARN: yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing (SoCC '13)*, pp. 5:1–5:16, ACM, Santa Clara, Calif, USA, 2013.

[3] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: what it is, and what it is not," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC '13)*, pp. 23–27, USENIX, San Jose, Calif, USA, 2013.

[4] K. J. Astrom and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*, Princeton University Press, Princeton, NJ, USA, 2008.

[5] M. Armbrust, A. Fox, R. Griffith et al., "Above the clouds: a berkeley view of cloud computing," Tech. Rep., University of California, Berkeley, Calif, USA, 2009.

[6] J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, and G. Antoniu, "GMonE: a complete approach to cloud monitoring," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2026–2040, 2013.

[7] A. Verma, B. Cho, N. Zea, I. Gupta, and R. H. Campbell, "Breaking the MapReduce stage barrier," *Cluster Computing*, vol. 16, no. 1, pp. 191–206, 2013.

[8] T.-C. Huang, K.-C. Chu, W.-T. Lee, and Y.-S. Ho, "Adaptive combiner for MapReduce on cloud computing," *Cluster Computing*, vol. 17, no. 4, pp. 1231–1252, 2014.

[9] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys*, vol. 42, no. 3, article 10, 2010.

[10] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta, "Making cloud intermediate data fault-tolerant," in *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*, pp. 181–192, ACM, New York, NY, USA, June 2010.

[11] F. Dinu and T. S. Eugene Ng, "Understanding the effects and implications of compute node related failures in Hadoop," in *Proceedings of the 21st ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC '12)*, pp. 187–197, Delft, The Netherlands, June 2012.

[12] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs scale-out for hadoop: time to rethink?" in *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC '13)*, pp. 20.1–20.13, ACM, New York, NY, USA, 2013.

[13] G. Ananthanarayanan, A. Ghodsi, A. Wang et al., "PACMan: coordinated memory caching for parallel jobs," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI '12)*, p. 20, USENIX Association, Berkeley, Calif, USA, 2012.

[14] K. Elmeleegy, "Piranha: optimizing short jobs in hadoop," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 985–996, August 2013.

[15] T. White, *Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale*, O'Reilly, 3rd edition, 2012.

[16] Apache Software Foundation, *Apache Hadoop NextGen MapReduce (YARN)*, 2015, http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[17] Hortonworks, *Hortonworks Data Platform: Installing HDP Manually*, 2013.

[18] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th ACM EuroSys Conference on Computer Systems (EuroSys '10)*, pp. 265–278, ACM, Paris, France, April 2010.

[19] S. Babu, "Towards automatic optimization of MapReduce programs," in *Proceedings of the Proceedings of the 1st ACM Symposium on Cloud Computing*, pp. 137–142, ACM, New York, NY, USA, June 2010.

[20] E. Jahani, M. J. Cafarella, and C. Ré, "Automatic optimization for MapReduce programs," *Proceedings of the VLDB Endowment*, vol. 4, no. 6, pp. 385–396, 2011.

[21] Z. Tang, J. Zhou, K. Li, and R. Li, "A MapReduce task scheduling algorithm for deadline constraints," *Cluster Computing*, vol. 16, no. 4, pp. 651–662, 2013.

[22] Y. Guo, J. Rao, C. Jiang, and X. Zhou, "FlexSlot: moving hadoop into the cloud with flexible slot management," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*, pp. 959–969, IEEE Press, New Orleans, La, USA, November 2014.

[23] S. Tang, B.-S. Lee, and B. He, "DynamicMR: a dynamic slot allocation optimization framework for mapreduce clusters," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 333–347, 2014.

[24] M. Li, L. Zeng, S. Meng et al., "MRONLINE: mapReduce online performance tuning," in *Proceedings of the ACM 23rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '14)*, pp. 165–176, New York, NY, USA, 2014.

[25] B. Palanisamy, A. Singh, L. Liu, and B. Langston, "Cura: a cost-optimized model for MapReduce in a cloud," in *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS '13)*, pp. 1275–1286, IEEE, Boston, Mass, USA, May 2013.

[26] A. Clement, M. Kapritsos, M. Kapritsos et al., "Upright cluster services," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*, pp. 277–290, New York, NY, USA, 2009.

[27] P. Costa, M. Pasin, A. Bessani, and M. Correia, "Byzantine fault-tolerant MapReduce: faults are not just crashes," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM '11)*, pp. 17–24, IEEE Computer Society, Washington, DC, USA, 2011.

[28] M. Correia, P. Costa, M. Pasin, A. Bessani, F. Ramos, and P. Verissimo, "On the feasibility of byzantine fault-tolerant mapreduce in clouds-of-clouds," in *Proceedings of the 31st Symposium on Reliable Distributed Systems (SRDS '12)*, pp. 448–453, IEEE, Irvine, Calif, USA, October 2012.

[29] Z. Xiao and Y. Xiao, "Achieving accountable MapReduce in cloud computing," *Future Generation Computer Systems*, vol. 30, no. 1, pp. 1–13, 2014.

[30] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, "Hadoop high availability through metadata replication," in *Proceedings of the 1st International Workshop on Cloud Data Management (CloudDB '09)*, pp. 37–44, ACM, Hong Kong, November 2009.

[31] H. Lin, X. Ma, and W.-C. Feng, "Reliable MapReduce computing on opportunistic resources," *Cluster Computing*, vol. 15, no. 2, pp. 145–161, 2012.

[32] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See Spot Run: using spot instances for mapreduce workflows," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud '10)*, p. 7, USENIX Association, Berkeley, Calif, USA, 2010.

[33] H. Liu, "Cutting mapReduce cost with spot market," in *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing (HotCloud '11)*, p. 5, Berkeley, Calif, USA, 2011.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 5612039, 11 pages
http://dx.doi.org/10.1155/2016/5612039

Research Article

Virtual Machine Placement Algorithm for Both Energy-Awareness and SLA Violation Reduction in Cloud Data Centers

Zhou Zhou,[1] Zhigang Hu,[1] and Keqin Li[2]

[1]*School of Software, Central South University, Changsha 410083, China*
[2]*Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*

Correspondence should be addressed to Zhou Zhou; zhouzhou03201@126.com

Received 7 February 2016; Accepted 10 March 2016

Academic Editor: Laurence T. Yang

The problem of high energy consumption is becoming more and more serious due to the construction of large-scale cloud data centers. In order to reduce the energy consumption and SLA violation, a new virtual machine (VM) placement algorithm named ATEA (adaptive three-threshold energy-aware algorithm), which takes good use of the historical data from resource usage by VMs, is presented. In ATEA, according to the load handled, data center hosts are divided into four classes: hosts with little load, hosts with light load, hosts with moderate load, and hosts with heavy load. ATEA migrates VMs on heavily loaded or little-loaded hosts to lightly loaded hosts, while the VMs on lightly loaded and moderately loaded hosts remain unchanged. Then, on the basis of ATEA, two kinds of adaptive three-threshold algorithm and three kinds of VMs selection policies are proposed. Finally, we verify the effectiveness of the proposed algorithms by CloudSim toolkit utilizing real-world workload. The experimental results show that the proposed algorithms efficiently reduce energy consumption and SLA violation.

1. Introduction

Cloud computing [1, 2] is derived from grid computing. At present, cloud computing is receiving more and more attention, through which people can access resources in a simple way. In contrast to previous paradigms, cloud computing provides infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS).

On one hand, the construction of a large-scale virtualized data centers meets the demand of computational power; on the other hand, such data centers consume a great many of electrical energy resources, leading to high energy consumption and carbon dioxide emissions. It has been reported that [3], in 2013, the total electricity consumption of global data center was more than 4.35 gigawatts, and annual growth rate was by 15%. The high energy consumption problem of virtualized data centers causes a series of problems, including energy wastes, low Return on the Investment (ROI), system instability, and more carbon dioxide emissions [4].

However, most hosts in data centers are in a state of low CPU utilization. Barroso and Hölzle [5] took a survey over half a year and found that most hosts in data centers operate at lower than 50% CPU utilization. Bohrer et al. [6] investigated the problem of high energy consumption and came to the same conclusion. Therefore, it is extremely necessary to reduce the energy consumption of data centers while keeping low SLA (Service Level Agreement) violation [7].

In this paper, we put forward a new VM deployment algorithm (ATEA), two kinds of adaptive three-threshold algorithm (KAM and KAI), and three kinds of VM selection policies to reduce energy consumption and SLA violation. We verify the effectiveness of the proposed algorithms through using the CloudSim toolkit.

The main contributions of the paper are summarized as follows:

(i) Proposing a novel VM deployment algorithm (ATEA). In ATEA, hosts in a data center are divided

into four classes according to their load. ATEA migrates VMs on heavily loaded or little-loaded hosts to lightly loaded hosts, while the VMs on lightly loaded and moderately loaded hosts remain unchanged.

(ii) Presenting two kinds of adaptive three-threshold algorithm to determine the three thresholds.

(iii) Putting forward three kinds of VMs selection policies and making three paired t-tests.

(iv) Evaluating the proposed algorithms by extensive simulation using the CloudSim toolkit.

The rest of this paper is organized as follows. In Section 2, the related work is discussed. Section 3 presents the power model, cost of VM migration, SLA violation metrics, and energy efficiency metrics. Section 4 proposes ATEA, two kinds of adaptive three-threshold algorithm, VM selection policy, and VM deployment algorithm. Experiments and performance evaluation are presented in Section 5. Section 6 concludes the paper.

2. Related Work

At present, there are various studies focusing on energy efficient resource management in cloud data centers. Constraint energy consumption algorithm [8–10] and energy efficiency algorithm [11–15] are two main types of algorithms for solving the problem of high energy consumption in data centers. The main idea of the constraint energy consumption algorithm is to reduce the energy consumption in data centers, but this type of algorithm focuses a little on (does not consider) the SLA violation. For example, Lee and Zomaya [8] proposed two heuristic algorithms (ECS, ECS + idle) to decrease the energy consumption, but the two algorithms are easy to fall into local optimum and do not consider the SLA violation. Hanson et al. [9] presented Dynamic Voltage and Frequency Scaling (DVFS) policy to save power in data centers. When the task number is large, DVFS policy raises the voltage of the processor in order to deal with the task in time; when the task number is small, DVFS policy decreases the voltage of processor for the purpose of saving power. Kang and Ranka [10] put forward an energy-saving algorithm, and they supposed that overestimated or underestimated execution time of tasks is bad for energy-saving algorithm. For overestimation, the extra available time should be assigned to other tasks in order to reduce energy consumption. Similarly, this energy-saving algorithm does not consider the SLA violation. Therefore, the constraint energy consumption algorithm does not meet the requirement of users because of focusing a little on (not considering) the SLA violation.

The goal of the energy efficiency (energy consumption and SLA violation) algorithm is to decrease the energy consumption and SLA violation in data centers. For example, Buyya et al. [11] raised a virtual machine (VM) placement algorithm (called Single Threshold (ST)) based on the combination of VM selection policies. ST algorithm sets a unified value for all servers' CPU utilization to make sure all servers are below this value. Obviously, ST algorithm can save energy consumption and decrease the SLA violation, but the SLA violation remains at a high level. Beloglazov and Buyya [12] proposed an energy efficient resource management system, which includes the dispatcher, global manager, local manager, and VMM (VM Monitor). In order to improve the energy efficiency, Beloglazov et al. put forward a new VM migration algorithm called Double Threshold (DT) [13]; DT sets two thresholds to keep all hosts' CPU utilization between the two thresholds. However, the energy consumption and SLA violation for DT algorithm need to be further decreased. Later, Beloglazov and Buyya [14, 15] proposed an adaptive double-threshold VM placement algorithm to improve energy efficiency in data centers. However, the energy consumption in data centers remains at a high level.

In our previous study [16], we proposed a three-threshold energy-aware algorithm named MIMT to deal with the energy consumption and SLA violation. However, the three thresholds for controlling host's CPU utilization are fixed. Therefore, MIMT is not suitable for varying workload. Therefore, it is necessary to put forward a novel VM placement algorithm to deal with energy consumption and SLA violation in cloud data centers.

3. The Power Model, Cost of VM Migration, SLA Violation Metrics, and Energy Efficiency Metrics

3.1. The Power Model. Energy consumption by servers in data centers is connected with its CPU, memory, disk, and bandwidth. Recent studies [17, 18] have illustrated that the energy consumption by servers has a linear relationship with its CPU utilization; even DVFS policy is applied. However, with the decrease of hardware price, multicore CPUs and memory with large-capacity are widely equipped in servers, leading to the traditional linear model not being able to accurately describe energy consumption of servers. In order to deal with this problem, we use the real data of energy consumption offered by SPECpower benchmark (http://www.spec.org/power_ssj2008/).

We have chosen two servers equipped with dual-core CPUs. The main configuration of the two servers is as follows: one is HP ProLiant G4 equipped with 1.86 GHz (dual-core), 4 GB RAM; the other is HP ProLiant G5 equipped with 2.66 GHz (dual-core), 4 GB RAM. The energy consumption for the two servers at different load levels is listed in Table 1 [15].

3.2. Cost of VM Migration. Proper VM migration between servers can reduce energy consumption and SLA violation in data centers. However, excessive VM migration could bring negative impact on performance of application which runs on the VMs. Voorsluys et al. [19] investigated the cost problem of VM migration, and the performance degradation caused by VM can be described in

$$C = k \cdot \int_{t_0}^{t_0+T_{m_j}} u_j(t)\, dt,$$
$$T_{m_j} = \frac{M_j}{B_j}, \tag{1}$$

Table 1: Power consumption by the two servers at different load levels in Watts.

Server	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP G5	93.7	97	101	105	110	116	121	125	129	133	135

where parameter C represents total performance degradation caused by VM j, parameter k is the coefficient of average performance degradation caused by VMs (in terms of the class of web-applications, the value of k could be estimated as approximately 0.1 (10%) of the CPU utilization [19]), function $u_j(t)$ corresponds to the CPU utilization of VM j, parameter t_0 represents start time of migration, T_{m_j} is completion time, M_j corresponds to the total memory used by VM j, and B_j represents the available bandwidth.

3.3. SLA Violation Metrics. SLA violation is extremely important factor for any VM migration algorithm. At present, there are two ways to describe the SLA violations.

(1) PDM [15] (Overall Performance Degradation Caused by VM Migration). It is indicated in

$$\text{PDM} = \frac{1}{M}\sum_{j=1}^{M}\frac{C_{d_j}}{C_{r_j}}, \tag{2}$$

where parameter M represents the number of VMs in data center, C_{d_j} means the estimate of the performance degradation caused by VM j migration, and C_{r_j} corresponds to the total CPU capacity requested by VM j during its lifetime.

(2) SLATAH [15] (SLA Violation Time per Active Host). It means the percentage of total SLA violation time, during which active host's CPU utilization has experienced 100%, as indicated in

$$\text{SLATAH} = \frac{1}{N}\sum_{i=1}^{N}\frac{T_{s_i}}{T_{a_i}}, \tag{3}$$

where N represents the number of hosts in data center, T_{s_i} corresponds to the total time, during which the CPU utilization of host i has experienced 100% utilization resulting in the SLA violations, T_{a_i} corresponds to the total time of host i being in active state. The reasoning behind the SLATAH is that the active host's CPU utilization has experienced 100% utilization, the VMs on the host could not be provided with the requested CPU capacity.

Both PDM and SLATAH are two effective methods to independently evaluate the SLA violation. Therefore, the SLA violation is defined as in [15]

$$\text{SLA} = \text{PDM} \times \text{SLATAH}. \tag{4}$$

3.4. Energy Efficiency Metric. Energy efficiency includes energy consumption and SLA violation. Improving the energy efficiency means less energy consumption and SLA violation in data centers. Therefore, the metric of energy efficiency is defined as in

$$E = \frac{1}{P \times \text{SLA}}, \tag{5}$$

where E corresponds to the energy efficiency of a data center, P means the energy consumption of a data center, and SLA represents the SLA violation of a data center. Equation (5) shows that the higher the E, the more the energy efficiency.

4. ATEA, Two Kinds of Adaptive Three-Threshold Algorithm, VM Selection Policy, and VM Deployment Algorithm

4.1. ATEA. VM migration is an effective method to improve the energy efficiency in data centers. However, there are several key problems which should be dealt with: (1) when a host is supposed to be heavily loaded, where some VMs from the host should be migrated to another host; (2) when a host is believed to be moderately loaded or lightly loaded, resulting in a decision to keep all VMs on this host unchanged; (3) when a host is believed to be little-loaded, where all VMs on the host must be migrated to another host; (4) selecting a VM or more VMs that should be migrated from the heavily loaded; (5) finding a new host to accommodate VMs migrated from heavily loaded or little-loaded hosts.

In order to solve the above problems, ATEA (adaptive three-threshold energy-aware algorithm) is proposed. ATEA automatically sets three thresholds T_l, T_m, and T_h ($0 \le T_l < T_m < T_h \le 1$), leading to the data center hosts to be divided into four classes: hosts with little load, hosts with light load, hosts with moderate load, and hosts with heavy load. When CPU utilization of a host is less than or equal to T_l, the host is believed to be little-loaded. In order to save energy consumption, all VMs on little-loaded host must be migrated to another host with light load and the host is switched to sleep mode; when CPU utilization of a host is between T_l and T_m, the host is considered as being lightly loaded. In order to avoid high SLA violations, all VMs on lightly loaded host have to be kept unchanged; the reason is that excessive VMs migration leads to the performance degradation and high SLA violations; when CPU utilization of a host is between T_m and T_h, the host is believed to be moderately loaded; all VMs on moderately loaded host have to be kept unchanged for the reason that excessive VMs migration leads to the performance degradation and high SLA violations; when CPU utilization of a host is greater than T_h, the host is considered as being heavily loaded; in order to reduce the SLA violations, some VMs on heavily loaded host must be migrated to another host with light load. Figure 1 shows the flow chart of algorithm ATEA.

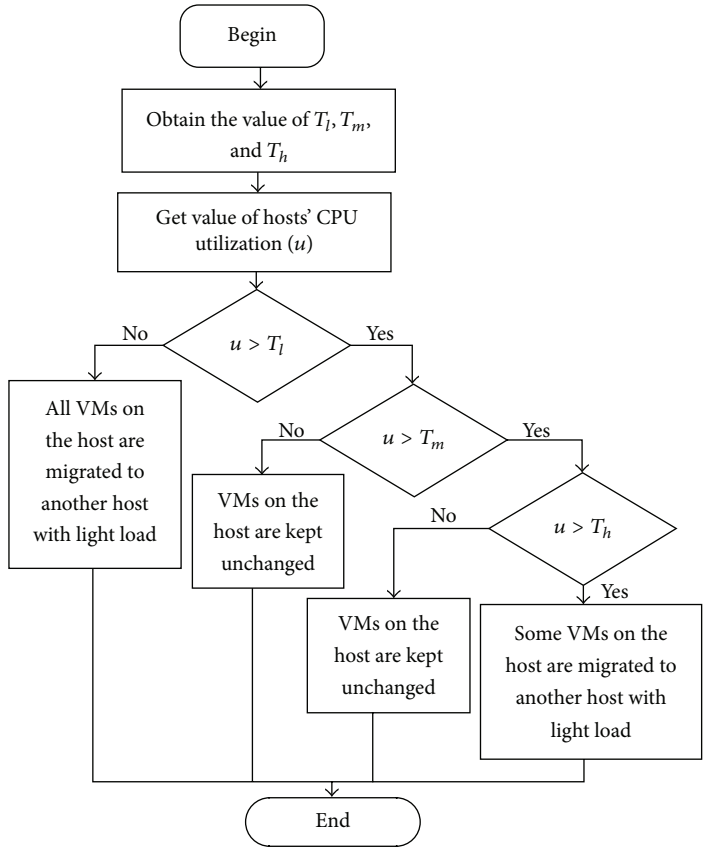


Figure 1: Flow chart of ATEA.

Different from out previous study [16], where the thresholds are fixed, in this paper, the three thresholds T_l, T_m, and T_h in ATEA are not fixed and these values can be autoadjusted according to workload.

ATEA migrates VMs that must be migrated to other hosts, while keeping some VMs unchanged. In doing so, ATEA improves the migration efficiency of VMs. Therefore, ATEA is a fine-grained algorithm. However, two problems should be solved as for ATEA. Firstly, what are the threshold values of T_l, T_m, and T_h? This problem will be discussed in Section 4.2. Secondly, as mentioned above, some VMs on heavily loaded host must be migrated to another host with light load. Which VM should be migrated? This issue will be discussed in Section 4.3.

The VM placement optimization of ATEA is illustrated in Algorithm 1.

In the first stage, the algorithm inspects each host in host list and decides which host is heavily loaded. If the host is heavily loaded (corresponding to Line 2 in Algorithm 1), the algorithm uses the VM selection policy to choose VMs which must be migrated from the host (corresponding to Line 6 in Algorithm 1). Once VMs list that should be migrated from the heavily loaded is created, the VM deployment algorithm is invoked for the purpose of finding a new host to accommodate the VM (corresponding to Line 7 in Algorithm 1). Function "getNewVmPlacement(vmsToMigrate)" means to find a new host to accommodate the VM. In the second stage, the algorithm inspects each host in host list and decides which host is little-loaded. If the host is little-loaded (corresponding to Line 11 in Algorithm 1), the algorithm chooses all VMs from the host to migrate and finds a placement of the VMs (corresponding to Line 15-Line 16 in Algorithm 1). At last, the algorithm returns the migration map.

4.2. Two Kinds of Adaptive Three-Threshold Algorithm. As discussed in Section 4.1, what are the threshold values of T_l, T_m, and T_h? To solve this problem, two adaptive three-threshold algorithms (KAM and KAI) are proposed.

4.2.1. KAM (K-Means Clustering Algorithm-Average-Median Absolute Deviation). For a univariate data set $V_1, V_2, V_3, \ldots, V_n$ (V_i is a host's CPU utilization at time i, and the size of n could be determined by empirical value), the KAM algorithm firstly uses the K-means clustering algorithm to divide the data set $(V_1, V_2, V_3, \ldots, V_n)$ into m groups $(G_1, G_2, \ldots, G_m)$ (the size of m could be obtained by empirical value, and in this paper, $m = 5$), where $G_k = (V_{j_{k-1}+1}, V_{j_{k-1}+2}, \ldots, V_{j_k})$, for all $1 \le k \le 5$, and $0 = j_0 < j_1 < j_2 < \cdots < j_5 = n$. Then, KAM gets the average value of each group, formalized as follows:

$$G_{Ak} = \frac{\left(V_{j_{k-1}+1} + V_{j_{k-1}+2} + \cdots + V_{j_k}\right)}{\left(j_k - j_{k-1}\right)}, \tag{6}$$

```
Require: hostlist              // hostlist is the set of all hosts
Ensure: migrationMap
(1) for each host in hostlist do
(2)     if isHostHeavilyLoaded(host) then
(3)         HeavilyLoadedHosts.add(host);    // host is heavily-loaded
(4)     end if
(5) end for
(6) vmsToMigrate = getVmsFromHeavilyLoadedHosts(HeavilyLoadedHosts);
(7) migrationMap.addAll(getNewVmPlacement(vmsToMigrate));
(8) HeavilyLoadedHosts.clear( );
(9) vmsToMigrate.clear( );
(10) for each host in hostlist do
(11)    if isHostLittleLoaded(host) then
(12)        LittleLoadedHosts.add(host); // host is little-loaded
(13)    end if
(14) end for
(15) vmsToMigrate = getVmsToMigrateFromHosts(LittleLoadedHosts);
(16) migrationMap.addAll(getNewVmPlacement(vmsToMigrate));
(17) return migrationMap
```

Algorithm 1: Optimized allocation of VMs.

for all $1 \le k \le 5$. Then, KAM gets the Median Absolute Deviation (MAD) of $G(G_{A1}, G_{A2}, \ldots, G_{A5})$. Therefore, the MAD is defined as follows:

$$\text{MAD} = \text{median}_{Ap}\left(\left|G_{Ap} - \text{median}_{Aq}\left(G_{Aq}\right)\right|\right), \quad (7)$$

where $A1 \le Ap \le A5$ and $\text{median}_{Aq}(G_{Aq})$ is median value of G_{Aq}. Finally, the three thresholds (T_l, T_m, and T_h) in ATEA could be defined as follows:

$$T_l = 0.5\,(1 - r \times \text{MAD}), \quad (8)$$

$$T_m = 0.9\,(1 - r \times \text{MAD}), \quad (9)$$

$$T_h = 1 - r \times \text{MAD}, \quad (10)$$

where $r \in R^+$ represents a parameter of the algorithm that defines how aggressively the system consolidates VMs. For example, the higher r, the more energy consumption, but the less SLA violations caused by VMs consolidation. The complexity of KAM is $O(m \times n \times t)$, where m is the group number, n denotes the data size, and t is the iteration number.

As the value of V_i ($i = 1, 2, 3, \ldots, n$) varies from time to time, the value of T_l, T_m, and T_h are also variable. Therefore, KAM is an adaptive three-threshold algorithm. When the workloads are dynamic and unpredictable, as compared with a fixed threshold algorithm, KAM generates higher energy efficiency by setting the value of T_l, T_m, and T_h.

4.2.2. KAI (K-Means Clustering Algorithm-Average-Interquartile Range). KAI is another adaptive threshold algorithm. For a univariate data set $V_1, V_2, V_3, \ldots, V_n$ (V_i is a host's CPU utilization at time i, and the size of n could be determined by empirical value), the KAI algorithm firstly uses the K-means clustering algorithm to divide the data set $(V_1, V_2, V_3, \ldots, V_n)$ into m groups $(G_1, G_2, \ldots, G_m)$ (the size of m could be obtained by empirical value, and in this paper, $m = 5$), where $G_k = (V_{j_{k-1}+1}, V_{j_{k-1}+2}, \ldots, V_{j_k})$, for all $1 \le k \le 5$, and $0 = j_0 < j_1 < j_2 < \cdots < j_5 = n$. Then, KAI gets the average value of each group, formalized as follows:

$$G_{Ak} = \frac{\left(V_{j_{k-1}+1} + V_{j_{k-1}+2} + \cdots + V_{j_k}\right)}{\left(j_k - j_{k-1}\right)}, \quad (11)$$

for all $1 \le k \le 5$. Then, KAI gets the Interquartile Range (IR) of $G(G_{A1}, G_{A2}, \ldots, G_{A5})$. Therefore, the IR is defined as follows:

$$\text{IR} = Q_3 - Q_1, \quad (12)$$

where Q_3 is third quartiles of G and Q_1 is first quartiles of G. Finally, the three thresholds (T_l, T_m, and T_h) in ATEA can be defined as follows:

$$T_l = 0.5\,(1 - r \times \text{IR}), \quad (13)$$

$$T_m = 0.9\,(1 - r \times \text{IR}), \quad (14)$$

$$T_h = 1 - r \times \text{IR}, \quad (15)$$

where $r \in R^+$ represents a parameter of the algorithm that defines how aggressively the system consolidates VMs. For example, the higher r, the more energy consumption, but the less SLA violations caused by VMs consolidation. The complexity of KAI is $O(m \times n \times t)$, where m is the group number, n denotes the data size, and t is the iteration number.

As the value of V_i ($i = 1, 2, 3, \ldots, n$) varies from time to time, the values of T_l, T_m, and T_h are also variable. Therefore, KAI is an adaptive three-threshold algorithm. When the workloads are dynamic and unpredictable, as compared with fixed threshold algorithm, KAI generates higher energy efficiency by setting the value of T_l, T_m, and T_h.

4.3. VM Selection Policies. As described earlier in Section 4.1, some VMs on heavily loaded host must be migrated to another host with light load. Which VM should be migrated? In general, a host's CPU utilization and memory size affect its energy efficiency. Therefore, to solve this problem, three kinds of VM selection policies (MMS, LCU, and MPCM) are proposed in this section.

4.3.1. MMS (Minimum Memory Size) Policy. The migration time of a VM will change, depending on its different memory size. A VM with less memory size means less migration time under the same spare network bandwidth. For example, a VM with 16 GB memory may take 16 times' longer migration time than a VM with 1 GB memory. Clearly, selecting the VM with 16 GB memory or the VM with 1 GB memory greatly affects energy efficiency of data centers. Therefore, if a host is being heavily loaded, MMS policy selects a VM with the minimum memory size to migrate compared with the other VMs allocated to the host. MMS policy chooses a VM u that satisfies the following condition:

$$\text{RAM}(u) \leq \text{RAM}(v), \quad \forall v \in \text{VM}_i, \tag{16}$$

where VM_i means the set of VMs allocated to host i and $\text{RAM}(u)$ is the memory size currently utilized by the VM u.

4.3.2. LCU (Lowest CPU Utilization) Policy. As for energy efficiency in data center, the CPU utilization of a host is also another important factor. Therefore, if a host is being heavily loaded, LCU policy chooses a VM with the lowest CPU utilization to migrate compared with the other VMs allocated to the host. LCU policy chooses a VM u that satisfies the following condition:

$$\text{Utilization}(u) \leq \text{Utilization}(v), \quad \forall v \in \text{VM}_i, \tag{17}$$

where VM_i means the set of VMs allocated to host i and $\text{Utilization}(u)$ is the CPU utilization of VM u allocated to host i.

4.3.3. MPCM (Minimum Product of Both CPU Utilization and Memory Size) Policy. As host's CPU utilization and memory size are two important factors for energy efficiency in data center, if a host is being heavily loaded, MPCM policy selects a VM with the minimum product of both CPU utilization and memory size to migrate compared with the other VMs allocated to the host. MPCM policy chooses a VM u that satisfies the following condition:

$$\left(u_{\text{CPU}} \times u_{\text{memory}}\right) \leq \left(v_{\text{CPU}} \times v_{\text{memory}}\right), \quad \forall v \in \text{VM}_i, \tag{18}$$

where VM_i means the set of VMs allocated to host i and u_{CPU} and u_{memory}, respectively, represent CPU utilization and memory size currently utilized by the VM u.

4.4. VM Deployment Algorithm. VM deployment can be considered as a bin packing problem. In order to tackle the problem, a modification of the Best Fit Decreasing algorithm denoted by Energy-Aware Best Fit Decreasing (EBFD) could be used to deal with it. As described earlier in Section 4.1, VMs on heavily loaded host or VMs on little-loaded host must be migrated to another host with light load (CPU utilization of a host at T_l-T_m interval); VMs on lightly loaded host or VMs on moderately loaded host are kept unchanged.

Algorithm 2 shows the pseudocode of EBFD, where T_l, T_m, and T_h are three thresholds in ATEA (the definition of the three thresholds in Section 4.2). "vmlist" is the set of all VMs. "hostlist" represents all hosts in the data centers. Line 1 (see Algorithm 2) means to sort all VM by CPU utilization in descending order. Line 3 represents that parameter "minimumPower" is assigned a maximum value. Line 6 is to check whether the host is suitable to accommodate the VM (e.g., host's CPU capacity, memory size, and bandwidth). Function getUtilizationAfterAllocation means to obtain host's CPU utilization after allocating a VM. Line 7 to Line 9 (see Algorithm 2) are to keep a host with light load (CPU utilization of a host at T_l-T_m interval). Function getPowerAfterVM is to obtain the increasing of host's energy consumption after allocating a VM. Line 11 to Line 15 are to find the host which owns the least increasing of power consumption caused by VM allocation. Line 19 is to return the destination hosts for accommodating VMs. The complexity of the EBFD is $O(m \times n)$; parameter m represents the number of hosts, whereas parameter n corresponds to the number of VMs which should be allocated.

5. Experiments and Performance Evaluation

5.1. Experiment Setup. Due to the advantages of CloudSim toolkit [20, 21] such as supporting on demand dynamic resource provisioning and modeling of virtualized environments and so on, we choose it as the simulation toolkit for our experiments.

We have simulated a data center which includes 800 heterogeneous physical nodes, half of which consists of HP ProLiant G4 (Intel Xeon 3040, 2 cores 1860 MHz, 4 GB), and the other half are HP ProLiant G5 (Intel Xeon 3075, 2 cores 2660 MHz, 4 GB). There are 1052 VMs and four kinds of VM types (High-CPU Medium Instance (2500 MIPS, 0.85 GB); Extra Large Instance (2000 MIPS, 3.75 GB); Small Instance (1000 MIPS, 1.7 GB); and Micro Instance (500 MIPS, 613 MB)) in the data center. The characteristics of the VMs correspond to Amazon EC2 instance types.

5.2. Workload Data. Using real workload to do experiments is extremely significant for VM placement. In this paper, we utilize the workload coming from a CoMon project, which mainly monitors infrastructure for PlanetLab [22]. We obtain the data derived from more than a thousand VMs' CPU utilization and the VMs placed at more than 500 places across the world. Table 2 [15] shows the characteristics of the data.

5.3. Experimental Results and Analysis. By using the workload data (described in Section 5.2), two kinds of adaptive three-threshold algorithm (KAM and KAI) and three kinds of VM selection policies (MMS, LCU, and MPCM) are simulated. In addition, for the two adaptive three-threshold

```
Require:  T_l, T_m, T_h, vmlist, hostlist
Ensure:   migrationMap
(1)  vmList.sortByCpuUtilization( );
     // sorted by CPU utilizatioin in descending order
(2)  for each vm in vmlist do
(3)     minimumPower = maximum;
        // minimunPower is assigned a maximum value
(4)     allocatedHost = null;
(5)     for each host in hostlist do
(6)        if (host is Suitable for Vm (vm)) then
(7)           utilization = getUtilizationAfterAllocation(host, vm);
(8)           if ((utilization < T_l) || (utilization > T_m)) then
(9)              continue;
(10)          end if
(11)          EnergyConsumption = getPowerAfterVM(host, vm);
(12)          if (EnergyConsumption < minimumPower) then
(13)             minimumPower = EnergyConsumption;
(14)             allocatedHost = host;
(15)          end if
(16)       end if
(17)    end for
(18) end for
(19) return allocationHost.
```

Algorithm 2: Energy-Aware Best Fit Decreasing (EBFD).

Table 2: Workload data characteristics (CPU utilization).

Date	Number of VMs	Mean	St. dev.	Quartile 1	Median	Quartile 3
03/03/2011	1052	12.31%	17.09%	2%	6%	15%

algorithm we have varied the parameters (r (7)–(9)) and ((11)–(13)) form 0.5 to 3.0 increasing by 0.5. Therefore, these variations have led to 36 combinations of the algorithms and parameters. For example, for KAM, there are three VM selection policies (MMS, LCU, and MPCM) denoted by KAM-MMS, KAM-LCU, and KAM-MPCM, respectively. Similarly, for KAI, there are also three VM selection policies (MMS, LCU, and MPCM) denoted by KAI-MMS, KAI-LCU, and KAI-MPCM, respectively. In the following section, we will discuss the energy consumption, SLATAH, PDM, SLA violations, and energy efficiency for the algorithms with different parameter.

(1) Energy Consumption. For the six algorithms (KAM-MMS, KAM-LCU, KAM-MPCM, KAI-MMS, KAI-LCU, and KAI-MPCM) with different parameter (0.5 to 3.0 increasing by 0.5), the energy consumption is shown in Figure 2, which shows the energy consumption for the six algorithms. As for KAM-MMS, KAM-LCU, and KAM-MPCM, KAM-MPCM leads to the least energy consumption, KAM-LCU the second energy consumption, and KAM-MMS the most energy consumption. The reason is that KAM-MPCM considers both CPU utilization and memory size when a host is with heavy load. Compared with KAM-MMS, KAM-LCU leads to less energy consumption. This can be explained by the fact that the processor (CPU) of a host consumes much more energy than its memory. Similarly, as for KAI-MMS, KAI-LCU, and KAI-MPCM, KAI-MPCM leads to the least energy consumption, KAI-LCU the second energy consumption, and KAI-MMS the most energy consumption. The reason is that KAI-MPCM considers both CPU utilization and memory size when a host is with heavy load. Compared with

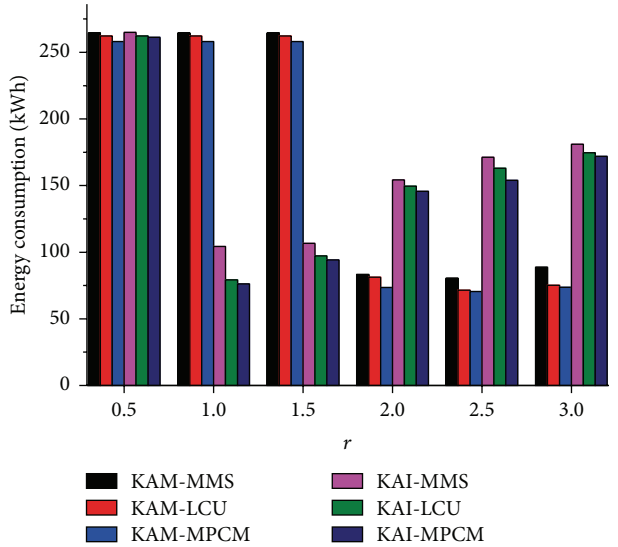


Figure 2: The energy consumption of the six algorithms.

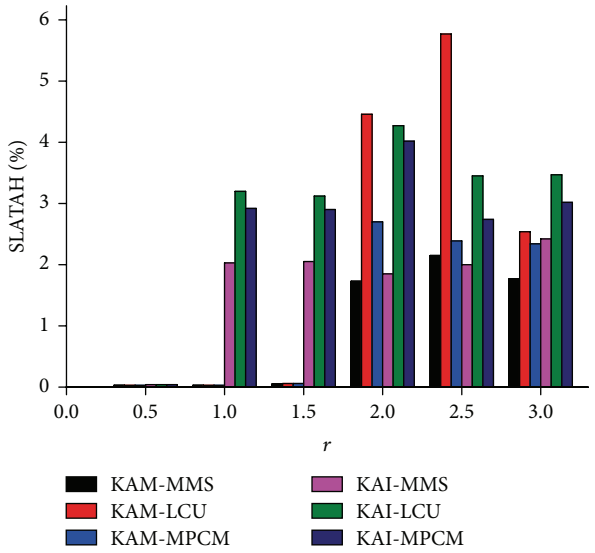


Figure 3: The SLATAH of the six algorithms.

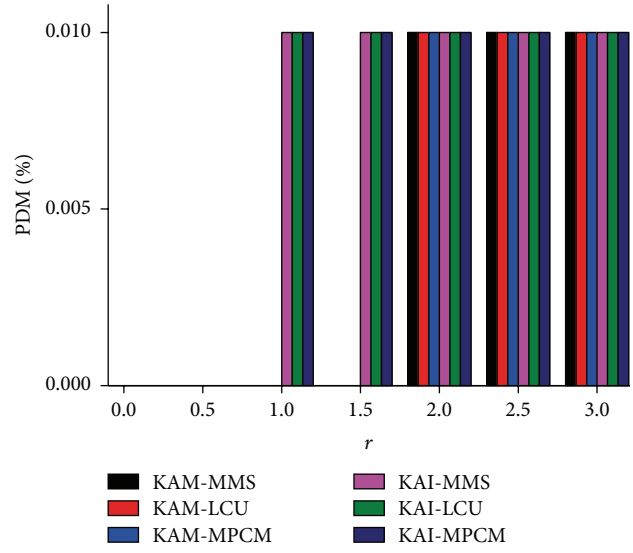


Figure 4: The PDM of the six algorithms.

KAI-MMS, KAI-LCU leads to less energy consumption. This can be also explained by the fact that the processor (CPU) of a host consumes much more energy than its memory.

(2) SLATAH. The SLATAH is described in Section 3.3 (see (3)). For the six algorithms (KAM-MMS, KAM-LCU, KAM-MPCM, KAI-MMS, KAI-LCU, and KAI-MPCM) with different parameter (0.5 to 3.0 increasing by 0.5), the SLATAH is shown in Figure 3, which shows the SLATAH for the six algorithms. In terms of KAM-MMS, KAM-LCU, and KAM-MPCM, KAM-MMS contributes to the least SLATAH, KAM-MPCM the second, and KAM-LCU the most. The reason is as follows: when a host is with heavy load, KAM-MMS selects a VM with the minimum memory size to migrate leading to less migration time. Therefore, KAM-MMS leads to the least SLATAH compared with KAM-LCU and KAM-MPCM. Compared with KAM-MPCM, KAM-LCU contributes to much more SLATAH. This could be explained by the fact that SLATAH mainly depends on memory size but not CPU utilization. Similarly, as for KAI-MMS, KAI-LCU, and KAI-MPCM, KAI-MMS contributes to the least SLATAH, KAI-MPCM the second, and KAI-LCU the most. The reason is that KAI-MMS causes the least migration time leading to the least SLATAH. Compared with KAI-MPCM, KAI-LCU leads to much more SLATAH. This could also be explained by the fact that SLATAH mainly depends on memory size but not CPU utilization.

(3) PDM. The PDM is described in Section 3.3 (see (2)). For the six algorithms (KAM-MMS, KAM-LCU, KAM-MPCM, KAI-MMS, KAI-LCU, and KAI-MPCM) with different parameter (0.5 to 3.0 increasing by 0.5), the PDM is shown in Figure 4, which illustrates the PDM for the six algorithms. As for KAM-MMS, KAM-LCU, and KAM-MPCM, the PDM is the same. This could be explained by the fact that the overall performance degradation caused by VMs due to migration is the same. Furthermore, when parameter $r = 0.5$, 1.0, and 1.5, respectively, the corresponding PDM of the three algorithms (KAM-MMS, KAM-LCU, and KAM-MPCM) are 0. As for KAI-MMS, KAI-LCU, and KAI-MPCM, the PDM is also the same. This could also be explained by the fact that the overall performance degradation caused by VMs due to migration is the same. At the same time, when parameter $r = 0.5$, the PDM of the three algorithms (KAI-MMS, KAI-LCU, and KAI-MPCM) are 0.

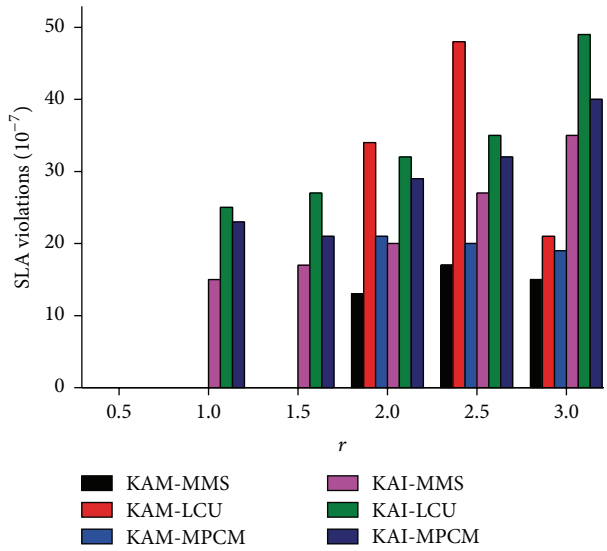


Figure 5: The SLA violations of the six algorithms.

(4) SLA Violations. The SLA violations are described in Section 3.3 (see (4)). For the six algorithms (KAM-MMS, KAM-LCU, KAM-MPCM, KAI-MMS, KAI-LCU, and KAI-MPCM) with different parameter (0.5 to 3.0 increasing by 0.5), the SLA violations are shown in Figure 5, which shows the SLA violations for the six algorithms. From (4), the SLA

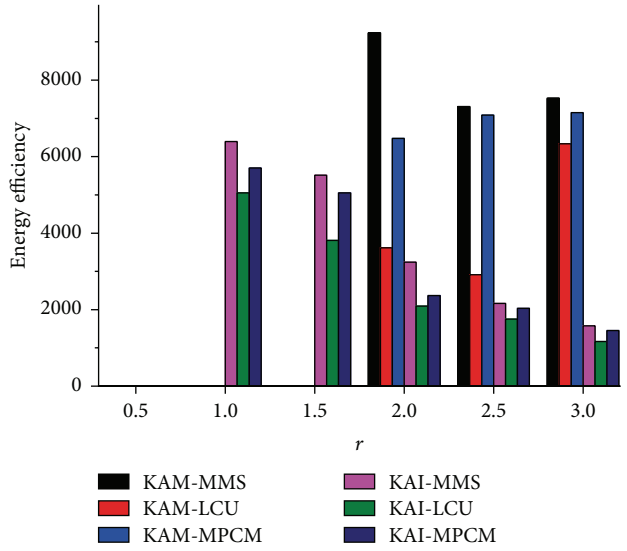


Figure 6: The energy efficiency of the six algorithms.

violations depend on SLATAH (Figure 3) and PDM (Figure 4). As for KAM-MMS, KAM-LCU, and KAM-MPCM, the PDM is the same. Therefore, SLA violations depend on SLATAH. In terms of KAM-MMS, KAM-LCU, and KAM-MPCM, KAM-MMS contributes to the least SLATAH, KAM-MPCM the second, and KAM-LCU the most. So, KAM-MMS contributes to the least SLA violations, KAM-MPCM the second, and KAM-LCU the most. Furthermore, when parameter $r = 0.5$, 1.0, and 1.5, respectively, the corresponding PDM of the three algorithms (KAM-MMS, KAM-LCU, and KAM-MPCM) are 0. Therefore, the corresponding SLA violations of the three algorithms (KAM-MMS, KAM-LCU, and KAM-MPCM) are 0. By the same way, as for KAI-MMS, KAI-LCU, and KAI-MPCM, KAI-MMS contributes to the least SLA violations, KAI-MPCM the second, and KAI-LCU the most. At the same time, when parameter $r = 0.5$, the PDM of the three algorithms (KAI-MMS, KAI-LCU, and KAI-MPCM) are 0. Therefore, the SLA violations of the three algorithms (KAI-MMS, KAI-LCU, and KAI-MPCM) are 0.

(5) Energy Efficiency. For the six algorithms (KAM-MMS, KAM-LCU, KAM-MPCM, KAI-MMS, KAI-LCU, and KAI-MPCM) with different parameter (0.5 to 3.0 increasing by 0.5), the energy efficiency (E) is shown in Figure 6, which shows the energy efficiency of the six algorithms. As discussed in Section 3.4, (5) depends on energy consumption (Figure 2) and SLA violation (Figure 5). Compared with KAM-LCU and KAM-MPCM, the energy efficiency of KAM-MMS is the most. The reason is that KAM-MMS reduces the migration time of VMs. In terms of energy efficiency, KAM-MPCM is better than KAM-LCU. This can be explained by the fact that KAM-MPCM considers both CPU utilization and memory size. As (5) is related to energy consumption (Figure 2) and SLA violation (Figure 5), when parameter $r = 0.5$, 1.0, and 1.5, respectively, the corresponding SLA violations of the three algorithms (KAM-MMS, KAM-LCU, and KAM-MPCM) are 0. Therefore, the corresponding energy efficiency of the three algorithms (KAM-MMS, KAM-LCU, and KAM-MPCM) is 0. Similarly, compared with KAI-LCU and KAI-MPCM, the energy efficiency of KAI-MMS is the most; the reason is that KAI-MMS reduces the migration time of VMs. In terms of energy efficiency, KAI-MPCM is better than KAI-LCU. This can be explained by the fact that KAI-MPCM considers both CPU utilization and memory size. As (5) is related to energy consumption (Figure 2) and SLA violation (Figure 5), when parameter $r = 0.5$, the corresponding SLA violations of the three algorithms (KAI-MMS, KAI-LCU, and KAI-MPCM) are 0. Therefore, the corresponding energy efficiency of the three algorithms (KAI-MMS, KAI-LCU, and KAI-MPCM) is 0.

Table 3: Comparison of the VMs select policies using paired t-tests.

Policy 1	Policy 2	Difference	P value
MMS (3579.5)	LCU (2228.3)	1531.2	P value < 0.05
MMS (3579.5)	MPCM (3109.9)	469.6	P value > 0.05
MPCM (3109.9)	LCU (2228.3)	881.6	P value < 0.05

Considering energy efficiency, we choose the parameter of six algorithms to maximize the energy efficiency. Figure 6 illustrates that parameter $r = 2.0$ for KAM-MMS is the best (denoted by KAM-MMS-2.0), parameter $r = 3.0$ for KAM-LCU is the best (denoted by KAM-LCU-3.0), parameter $r = 3.0$ for KAM-MPCM is the best (denoted by KAM-MPCM-3.0), parameter $r = 1.0$ for KAI-MMS is the best (denoted by KAI-MMS-1.0), parameter $r = 1.0$ for KAI-LCU is the best (denoted by KAI-LCU-1.0), and parameter $r = 1.0$ for KAI-MPCM is the best (denoted by KAI-MPCM-1.0).

For the two kinds of adaptive three-threshold algorithms (KAM and KAI), there are three VMs select policies which could be provided. Does a policy that is the best compared with other two policies in regard to energy efficiency exist? If it exists, which VM select policy is the best? In order to solve this problem, we have made three paired t-tests to determine which VM policy is the best in regard to energy efficiency. Before using the three paired t-tests, according to Ryan-Joiner's normality test, we verify the three VM select policies (MMS, LCU, and MPCM) with different parameter (r) that maximization energy efficiency follows a normal distribution with P value = 0.2 > 0.05. The paired t-tests results are showed in Table 3, which shows that MMS leads to a statistically significantly upper value of energy efficiency with P value < 0.05 compared with LCU and MPCM. In other words, in terms of energy efficiency, MMS is the best VM select policy compared with LCU and MPCM. Therefore, KAM-MMS-2.0 and KAI-MMS-1.0 are the best combination in regard to energy efficiency. In the following section, we use KAM-MMS-2.0 and KAI-MMS-1.0 to make comparison with other energy-saving algorithms.

(6) Comparison with Other Energy-Saving Algorithms. In this section, the NPA (Nonpower Aware), DVFS [9], THR-MMT-1.0 [15], THR-MMT-0.8 [15], MAD-MMT-2.5 [15], IQR-MMT-1.5 [15], and MIMT [16] are chosen to make comparison in regard to energy efficiency. The related experimental results are shown in Table 4.

Table 4: The comparison of the energy-saving algorithms.

Algorithm	Energy efficiency (KWh)	Energy consumption ($\times 10^{-7}$)	SLA violations	SLATAH (%)	PDM (%)	Number of VM migrations
NPA	—	2410.8	—	—	—	—
DVFS	—	803.91	—	—	—	—
THR-MMT-1.0	38	99.95	2613	26.97	0.10	19852
THR-MMT-0.8	170	119.40	492	4.99	0.10	26567
MAD-MMT-2.5	169	114.27	518	5.24	0.10	25923
IQR-MMT-1.5	166	117.08	514	5.08	0.10	26420
MIMT (0–40%–80%)	5420	108.53	17	2.86	0.01	8418
MIMT (5%–45%–85%)	4949	112.26	18	3.22	0.01	8687
KAM-MMS-2.0	9231	83.33	13	1.73	0.01	6808
KAI-MMS-1.0	6393	104.28	15	2.03	0.01	7519

Table 4 illustrates the energy efficiency, energy consumption, and SLA violations for the energy-saving algorithms. As the NPA algorithm does not take any energy-saving policy leading to 2410.8 KWh, DVFS algorithm reduces this value to 803.91 KWh by adjusting its CPU frequency dynamically. Because NPA and DVFS algorithm have no VMs migration, the symbol "—" is used to represent the energy efficiency, SLA violations, SLATAH, PDM, and number of VMs' migration. In terms of energy efficiency, THR-MMT-1.0 and THR-MMT-0.8 algorithms are better than DVFS. The reason is that THR-MMT-1.0 and THR-MMT-0.8 algorithms set the fixed threshold to migrate VMs leading to upper energy efficiency.

MAD-MMT-2.5 and IQR-MMT-1.5 are two kinds of adaptive threshold algorithm, which (MAD-MMT-2.5 and IQR-MMT-1.5) are better than THR-MMT-1.0 in regard to energy efficiency. This could be described by the fact that MAD-MMT-2.5 and IQR-MMT-1.5 dynamically set two thresholds to improve the energy efficiency. But compared with THR-MMT-0.8, the three algorithms (MAD-MMT-2.5, IQR-MMT-1.5, and THR-MMT-0.8) are at the same level in regard to energy efficiency. This could be explained by the fact that the 0.8 threshold value is a proper value for the THR-MMT algorithm.

KAM-MMS-2.0 and KAI-MMS-1.0 algorithms are better than MIMT (0–40%–80%) and MIMT (5%–45%–85%). This could be explained by the fact that MIMT (0–40%–80%) and MIMT (5%–45%–85%) set the fixed threshold to control the migration of VMs, while KAM-MMS-2.0 and KAI-MMS-1.0 autoadjust threshold to control the migration of VMs according to the workload.

Table 4 also shows that KAM-MMS-2.0 and KAI-MMS-1.0 algorithms, respectively, improve the energy efficiency compared with IQR-MMT-1.5, MAD-MMT-2.5, THR-MMT-0.8, and THR-MMT-1.0 and maintain the low SLA violation of 13×10^{-7}. In addition, KAM-MMS-2.0 and KAI-MMS-1.0 algorithms, respectively, generate 2-3 times fewer VMs migrations than IQR-MMT-1.5, MAD-MMT-2.5, THR-MMT-0.8, and THR-MMT-1.0. The reason is as follows. On one hand, KAM-MMS-2.0 and KAI-MMS-1.0 are two kinds of adaptive three-threshold algorithm, which dynamically set three thresholds, leading to the data center hosts to be divided into four classes (hosts with little load, hosts with light load, hosts with moderate load, and hosts with heavy load), and migrate the VMs on heavily loaded and little-loaded hosts to the host with light load, while the VMs on lightly loaded and moderately loaded hosts remain unchanged. Therefore, KAM-MMS-2.0 and KAI-MMS-1.0 algorithm reduce the migration time and improve the migration efficiency compared with other energy-saving algorithms. On the other hand, KAM-MMS-2.0 and KAI-MMS-1.0, respectively, select the best VMs select policy (MMS) combination with the best parameter (r) compared with other energy-saving algorithms.

6. Conclusions

This paper proposes a novel VMs deployment algorithm based on the combination of adaptive three-threshold algorithm and VMs selection policies. This paper shows that dynamic thresholds are more energy efficient than fixed threshold. The proposed algorithms are expected to be applied in real-world cloud platforms, aiming at reducing the energy costs for cloud data centers.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by the National Natural Science Foundation (nos. 61572525 and 61272148), the Ph.D. Programs Foundation of Ministry of Education of China (no. 20120162110061), the Fundamental Research Funds for the Central Universities of Central South University (no. 2014zzts044), the Hunan Provincial Innovation Foundation for Postgraduate (no. CX2014B066), and China Scholarship Council.

References

[1] H. Khazaei, J. Mišić, V. B. Mišić, and S. Rashwand, "Analysis of a pool management scheme for cloud computing centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 5, pp. 849–861, 2013.

[2] J. Carretero and J. G. Blas, "Introduction to cloud computing: platforms and solutions," *Cluster Computing*, vol. 17, no. 4, pp. 1225–1229, 2014.

[3] L. Wang and S. U. Khan, "Review of performance metrics for green data centers: a taxonomy study," *Journal of Supercomputing*, vol. 63, no. 3, pp. 639–656, 2013.

[4] D. Rincón, A. Agustí-Torra, J. F. Botero et al., "A novel collaboration paradigm for reducing energy consumption and carbon dioxide emissions in data centres," *The Computer Journal*, vol. 56, no. 12, pp. 1518–1536, 2013.

[5] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[6] P. Bohrer, E. N. Elnozahy, T. Keller et al., "The case for power management in web servers," in *Power Aware Computing*, Computer Science, pp. 261–289, Springer, Berlin, Germany, 2002.

[7] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012–1023, 2013.

[8] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374–1381, 2011.

[9] H. Hanson, S. W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio, "Thermal response to DVFS: analysis with an Intel Pentium M," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '07)*, pp. 219–224, August 2007.

[10] J. Kang and S. Ranka, "Dynamic slack allocation algorithms for energy minimization on parallel machines," *Journal of Parallel and Distributed Computing*, vol. 70, no. 5, pp. 417–430, 2010.

[11] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: challenges and opportunities," in *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS '09)*, pp. 1–11, Leipzig, Germany, June 2009.

[12] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid '10)*, pp. 826–831, IEEE, Melbourne, Australia, May 2010.

[13] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[14] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proceedings of the ACM 8th International Workshop on Middleware for Grids, Clouds and e-Science (MGC '10)*, pp. 1–6, Bangalore, India, December 2010.

[15] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency Computation Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[16] Z. Zhou, Z.-G. Hu, T. Song, and J.-Y. Yu, "A novel virtual machine deployment algorithm with energy efficiency in cloud computing," *Journal of Central South University*, vol. 22, no. 3, pp. 974–983, 2015.

[17] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*, pp. 13–23, ACM, June 2007.

[18] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[19] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: a performance evaluation," in *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1–4, 2009. Proceedings*, vol. 5931 of *Lecture Notes in Computer Science*, pp. 254–265, Springer, Berlin, Germany, 2009.

[20] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[21] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "CloudAnalyst: a cloudsim-based visual modeller for analysing cloud computing environments and applications," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA '10)*, pp. 446–452, IEEE, Perth, Australia, April 2010.

[22] K. S. Park and V. S. Pai, "CoMon: a mostly-scalable monitoring system for PlanetLab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 9823213, 13 pages
http://dx.doi.org/10.1155/2016/9823213

Research Article

Energy-Efficient Reliability-Aware Scheduling Algorithm on Heterogeneous Systems

Xiaoyong Tang[1,2] and Weizhen Tan[3]

[1]*School of Information Science and Engineering, National Supercomputing Center in Changsha, Hunan University, Changsha 410082, China*
[2]*Information Science and Technology College/Southern Regional Collaborative Innovation Center for Grain and Oil Crops in China, Hunan Agricultural University, Changsha 410128, China*
[3]*Archive, Hunan University of Humanities, Science and Technology, Loudi 417000, China*

Correspondence should be addressed to Weizhen Tan; twb1022@163.com

Received 22 December 2015; Accepted 24 February 2016

Academic Editor: Florin Pop

The amount of energy needed to operate high-performance computing systems increases regularly since some years at a high pace, and the energy consumption has attracted a great deal of attention. Moreover, high energy consumption inevitably contains failures and reduces system reliability. However, there has been considerably less work of simultaneous management of system performance, reliability, and energy consumption on heterogeneous systems. In this paper, we first build the precedence-constrained parallel applications and energy consumption model. Then, we deduce the relation between reliability and processor frequencies and get their parameters approximation value by least squares curve fitting method. Thirdly, we establish a task execution reliability model and formulate this reliability and energy aware scheduling problem as a linear programming. Lastly, we propose a heuristic Reliability-Energy Aware Scheduling (REAS) algorithm to solve this problem, which can get good tradeoff among system performance, reliability, and energy consumption with lower complexity. Our extensive simulation performance evaluation study clearly demonstrates the tradeoff performance of our proposed heuristic algorithm.

1. Introduction

For a long time, energy consumption has simply been ignored in the performance evaluation in large-scale parallel computing systems. However, Intelligence (DCDi) Industry Census reported that the amount of electricity consumed by global data centers ran up to 40 GW in 2013, and it was also with a 7% increase [1]. According to the latest world's Top 500 supercomputers Ranking, the power consumption of first supercomputer "Tianhe-2" is 17.808 MW and average power consumption for Top 10 systems in Ranking list is 6.2939 MW, respectively [2]. Thus, it is obvious that high energy cost is a key feature of designing and applying heterogeneous systems.

On the other hand, computing systems are a group of heterogeneous processors connected *via* a high-speed network that supports the execution of parallel applications. For example, the Top supercomputer "Tianhe-2" in Top 500 lists consists of Intel Xeon® E5-2692 12C 2.200 GHz and Intel Xeon Phi 31S1P (MIC) [2]. For each processor, the number of transistors integrated into today's Intel Xeon EX processor reaches to nearly 2.3 billion and its power consumption over 130 W [3]. This implies the possibility of worsening single processor reliability, eventually resulting in poorness of the whole heterogeneous system reliability. Furthermore, the modern large-scale computing systems usually have a lot of processors, such as "Tianhe-2" with 3,120,000 cores and "Titan" with 560,640 cores [2]. One of the main problems existing in this situation is system reliability, which drastically decreases as the number of processor cores increases [4]. Even when the single processor's one-hour reliability becomes very high, such as 0.999999, as the system size approaches 10,000 cores, the system's MTTF (the Mean Time to Failure) drops to less than 10 hours [4]. This also allows

us to focus primarily on the main problem of this paper, which is the *simultaneous management of system performance, reliability, and energy consumption*.

In recognition of this, we first build a reliability and energy aware task scheduling architecture including precedence-constrained parallel applications and energy consumption model on heterogeneous systems. Then, we propose the single processor failure rate model based on DVFS technique and deduce the application reliability of systems. Finally, to provide an optimum solution for this problem, we propose a heuristic Reliability-Energy Aware Scheduling (REAS) algorithm, which adopts a novel scheduling objective RE. The overall objective of this paper is trying to get good tradeoff among performance, reliability, and energy consumption.

The rest of the paper is organized as follows: the related work is summarized in Section 2. We describe the task scheduling system model in Section 3. In Section 4, we provide a system reliability model. To solve this problem, a heuristic reliability and energy aware task scheduling algorithm is proposed in Section 5. In Section 6, we verify the performance of the proposed algorithm by comparing the results obtained from performance evaluation. Finally, we summarize the contributions and make some remarks on further research in Section 7.

2. Related Work

The high-performance parallel application running on computing systems is usually composed of intercommunicated tasks, which are scheduled to run over different processors in the systems. In most cases, the main objective of scheduling strategies is to map the multiple interacting program tasks onto processors and order their executions so that task precedence requirements are satisfied and, in the meanwhile, the minimum schedule length (makespan) can be achieved. The problem of finding the optimal schedule is NP-complete in general [5–9]. There are many scheduling algorithms that have been proposed to deal with this problem, for example, dynamic-level scheduling (DLS) algorithm [6] and heterogeneous earliest-finish-time (HEFT) algorithm [5, 8, 10, 11].

As the energy consumption has become important issue in designing large-scale computing systems in the last few years, many techniques including dynamic voltage-frequency scaling (DVFS), dynamic powering on/off, slack reclamation, resource hibernation, and memory optimizations have been investigated and developed to reduce energy consumption [12–14]. DVFS, which is a technique in which a processor runs at a less-than-maximum frequency when it is not fully utilized in order to conserve power, is perhaps the most appealing method for reducing energy consumption [14, 15]. Most of the early DVFS-enabled researches focused on the single processor of embedded and real-time computing systems [14, 16, 17]. Recently, there has been a significant amount of work on task scheduling for heterogeneous systems using DVFS-enabled techniques. For instance, Rountree et al. focused on energy optimization of MPI program in HPC environment and proposed a linear programming (LP), which incorporates allowable time delays, communication slack, and memory pressure into its scheduling using DVFS (i.e., slack reclamation) [18]. Rizvandi et al. proposed a method to find the best frequencies of processor to obtain the optimal energy consumption [19]. Lee and Zomaya addressed the problem of scheduling precedence-constrained parallel applications on multiprocessor computer systems and their scheduling decisions are made using the relative superiority metric (RS) devised as a novel objective function [20]. In [21], Zong et al. proposed two energy-efficient scheduling algorithms (EAD and PEBD) for parallel tasks on homogeneous clusters based on duplication strategy.

All of this work demonstrated that dynamic adjusting the processor's voltage and frequency can effectively reduce system energy consumption. However, recent researches have illustrated that scaling the processor's voltage and frequency has negative impact of nanoscale semiconductor circuits's cosmic ray radiations, electromagnetic interference, and alpha particles, which enforce the unreliability of processor [22–24]. Thus, it is a good way to incorporate the reliability into energy aware scheduling based on DVFS. Recently, Zhu etc. focused on reducing energy consumption while preserving the system reliability for periodic real-time tasks [25, 26]. They proposed a reliability model that the processor's reliability decreases as scaling their voltage and frequency from max to min and incorporated the reliability requirements into heuristic energy aware task scheduling strategies. However, their techniques are not suitable for precedence-constrained parallel applications on heterogeneous systems based on DVFS-enabled processors.

Many researches had dealt with the reliability on heterogeneous systems. For example, Dogan and Özgüner introduced three reliability cost functions that were incorporated into making dynamic level (DL) and proposed a reliable dynamic level scheduling algorithm (RDLS) [27]; the goal was to minimize not only the execution time but also the failure probability of the application. In our previous work [8], we propose a scheduling algorithm which considers the task's execution reliability. Qin and Jiang investigated a dynamic and reliability-cost-driven (DRCD) scheduling algorithms for precedence-constrained tasks in heterogeneous clusters [28]. Unfortunately, those works did not consider the energy consumption and the reliability of scaling the processor's voltage and frequency. In recognition of this, we focus on the reliability and energy consumption on DVFS-enabled heterogeneous systems.

3. System Models

3.1. Scheduling Architecture. Various task scheduling architectures are proposed in literature [5, 8, 9, 14, 28, 29]. However, the energy consumption and system reliability are not effectively incorporated into scheduling. In this paper, we propose a reliability and energy aware task scheduling architecture, as depicted in Figure 1(a). It is assumed that all parallel applications, along with information provided by user, are submitted to system by a special user command. First, the parallel applications are divided as a task DAG by *Task DAG Model*. Then, the

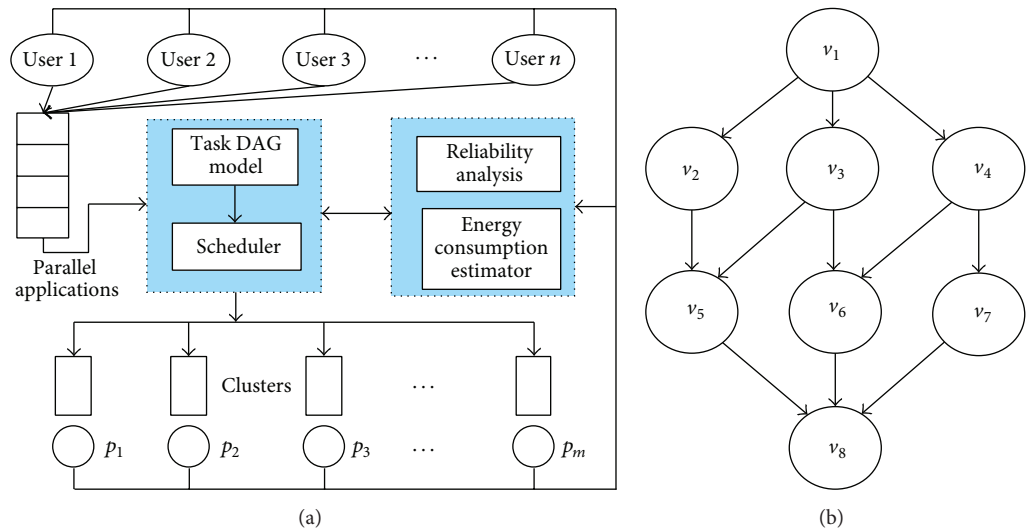


Figure 1: (a) The reliability and energy aware task scheduling architecture. (b) A parallel application task graph.

Table 1: The parameters of heterogeneous processors.

	λ_k	$P_s[k]$	$\alpha[k]$	$(f_{k,l}, V_{k,l})$		
				1	2	3
p_1	1.4×10^{-4}	73.6	3.663×10^{-8}	$(0.8 \times 10^9, 0.93)$	$(2.1 \times 10^9, 1.23)$	$(3.2 \times 10^9, 1.43)$
p_2	1.62×10^{-4}	57.1	4.95×10^{-8}	$(2.3 \times 10^9, 0.85)$	$(3.0 \times 10^9, 1.36)$	

estimate energy consumption of tasks, which are executed on the DVFS-enabled heterogonous processors, is computed by the *Eneregy Consumption Estimator*. At the same time, *reliability analysis* computes the processors' reliability according to different frequency to get the whole system reliability. Finally, the *Scheduler* schedules tasks based on the above task energy consumption and system reliability.

3.2. Heterogeneous Systems. The target system used in this work consists of a set of $P = \{p_1, p_2, \ldots, p_m\}$ heterogeneous processors/machines [5, 8, 9, 14, 29], which are connected by high-speed interconnects, such as Infiniband and Myrinet. Each DVFS-enabled processor $p_k \in P$ can adjust its operational voltage and frequency [14]. Therefore, they can be executed on discrete set of frequency-voltage pairs, $(f_{k,l}, V_{k,l})$, in which $(f_{k,1} < f_{k,2} < \cdots < f_{k,M_k})$ and $(V_{k,1}, V_{k,2} < \cdots < V_{k,M_k})$, where M_k is processor p_k's operation level [14, 30]. For example, the quad-core AMD Phenom II supports 4 different frequencies (0.8 GHz, 2.1 GHz, 2.5 GHz, and 3.2 GHz) and voltages ranging from 0.85 V to 1.425 V [30]. Since clock frequency transition overheads take a negligible amount of time (e.g., 10 us–150 us), these overheads are not considered in our study.

The heterogeneous processor's failure is assumed to follow a Poisson process and each processor has a constant failure rate λ [8, 9, 29]. For example, λ_k denotes a processor p_k failure rate when it works at normal voltage and frequency [8, 9, 27, 29]. These failure rates can be derived from system's profiling, system log, and statistical prediction techniques [31]. For demonstration purposes, we illustrate two heterogeneous processors, one has 3 frequency levels and the other has 2 frequency levels, and the parameters are listed in Table 1.

3.3. Applications Model. The precedence-constrained tasks of parallel application are usually denoted as a Directed Acyclic Graph (DAG) $G = \langle V, R, [d_{i,j}, w_{i,k,l}] \rangle$ [5, 8–10, 29], where $V = \{v_1, v_2, \ldots, v_n\}$ is the set with n tasks that can be scheduled to any available DVFS-enabled processors [5, 8–10, 29]; R represents the precedence relation that defines a partial order on the task set V, such that $v_i R v_j$ implies that the task v_i must be finished, before v_j can start execution [5, 8–10, 29]. $[d_{i,j}]$ is $n \times n$ communication matrix that denotes the communication time between tasks v_i and v_j for $1 \leq i, j \leq n$. $[w_{i,k,l}]$ is $n \times m \times M_{\max}$ computation matrix in which each $w_{i,k,l}$ gives the estimated time to execute task v_i on processor p_k at frequency $f_{k,l}$. Here, $M_{\max}$ is the maximal operation level on systems. The communication cost and computation cost can be evaluated by building a historic table and using code profiling or statistical prediction techniques [31]. Figure 1(b) shows a parallel application DAG, Table 2 lists the tasks execution time on two heterogeneous DVFS-enabled processors listed in Table 1, and the communication time among these tasks is listed in Table 3.

Generally, the common objective of task scheduling is to map tasks with precedence constrained onto processors and get a minimum schedule length (which is also called makespan) [10, 11]. Before presenting the schedule length, it is necessary to define the scheduling attributes EST and EFT

Table 2: Task estimated execution matrix $[w_{i,k,l}]$.

Task	p_1			p_2	
	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	$p_{2,1}$	$p_{2,2}$
v_1	11.12	2.28	2.87	3.89	3
v_2	36.29	13.82	9.12	12.7	9.78
v_3	15.46	5.91	3.92	5.45	4.21
v_4	5.33	2.01	1.4	1.94	1.49
v_5	66.77	25.44	16.77	23.28	17.83
v_6	13.82	5.3	3.53	4.84	3.75
v_7	7.43	2.86	1.89	2.68	2.04
v_8	8.48	3.19	2.09	2.91	2.31

Table 3: Estimated communication matrix $[d_{i,j}]$.

Task	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	6.99	15.48	6.69				
v_2				10.86			
v_3				1.25	12.56		
v_4					6.93	0.3	
v_5							0.11
v_6							6.535
v_7							6.2

of task v_i. $\mathrm{EST}(v_i, f_{k,l})$ denotes the earliest execution starting time of task $v_i \in V$ on DVFS-enabled processor $p_k \in P$ at frequency $f_{k,l}$, which is constrained by tasks precedence relation and the available time of processor p_k [5, 8–10, 29]. $\mathrm{EFT}(v_i, f_{k,l})$ is the earliest execution finish time of task v_i on processor p_k at frequency $f_{k,l}$, which is described as

$$\mathrm{EFT}\left(v_i, f_{k,l}\right) = \mathrm{EST}\left(v_i, f_{k,l}\right) + w_{i,k,l}. \tag{1}$$

In this paper, let $X_{i,k}^l = 1$ denote the task v_i scheduled on processor p_k at frequency $f_{k,l}$; otherwise $X_{i,k}^l = 0$. Thus, the schedule length is defined as follows:

$$\text{makespan} = \max_{1\le i\le n,\, 1\le k\le m}^{1\le l\le M_k} \left\{X_{i,k}^l \mathrm{EFT}\left(v_i, f_{k,l}\right)\right\}. \tag{2}$$

3.4. Energy Model. The major energy consumption of computing systems depends on its memory, disks, CPUs, and other components. This paper only considers DVFS-enabled CPUs, which consume the largest proportion of energy on systems [14, 19, 20, 32]. The power consumption of DVFS-enabled microprocessor based on complementary metal-oxide semiconductor (CMOS) logic circuits mainly consists of *static power* and *dynamic power dissipation*, which can be modeled as [25, 26]

$$P = P_s + \emptyset P_d, \tag{3}$$

where P_s is the *static power*, which is a constant and the power used to maintain basic circuits and keep the clock running, and *frequency-independent active power*. $\emptyset$ denotes the processor's model, if processor is at *execution* model, $\emptyset = 1$; otherwise, $\emptyset = 0$. P_d is the most significant factor of processor power consumption and can be estimated as [14, 16, 19, 20, 32]

$$P_d = \alpha V^2 f, \tag{4}$$

where α represents the switched capacitance, V is the supply voltage, f represents processor's working frequency, and σ stands for circuit dependent constant. The example of such processor parameters is listed in Table 1.

Let $\mathrm{EN}(v_i, f_{k,l})$ be the energy consumption caused by task v_i running on DVFS-enabled processor p_k at frequency $f_{k,l}$, of which it is determined by task execution time and processor power consumption:

$$\begin{aligned}\mathrm{EN}\left(v_i, f_{k,l}\right) &= w_{i,k,l} \times P^k \\ &= w_{i,k,l} \times P_s^k + w_{i,k,l} \times P_d\left(f_{k,l}\right),\end{aligned} \tag{5}$$

where $P_d(f_{k,l})$ denotes *dynamic power dissipation* of processor p_k at frequency $f_{k,l}$ (see (4)). Thus, for an application G, the energy consumption $\mathrm{EN}(V)$ is the summation of all tasks of energy consumption:

$$\begin{aligned}\mathrm{EN}\left(V\right) &= \sum_{1\le i\le n,\, 1\le k\le m}^{1\le l\le M_k} \left\{X_{i,k}^l \mathrm{EN}\left(v_i, f_{k,l}\right)\right\} \\ &= \sum_{1\le i\le n,\, 1\le k\le m}^{1\le l\le M_k} \left\{X_{i,k}^l w_{i,k,l} \times P_s^k + X_{i,k}^l w_{i,k,l} \times P_d\left(f_{k,l}\right)\right\}.\end{aligned} \tag{6}$$

At the same time, for heterogeneous systems, all processors are power-on; they are sleep or execution model. That is to say, all processors of systems consume *static power* all the time. Thus, the computing systems energy consumption $\mathrm{EN}(P)$ is the summation of all processors *static power* and *dynamic power dissipation* of application energy consumption:

$$\begin{aligned}\mathrm{EN}\left(P\right) = \text{makespan} \times &\sum_{k=1,2,\ldots,m} P_s^k \\ &+ \sum_{1\le i\le n,\, 1\le k\le m}^{1\le l\le M_k} \left\{X_{i,k}^l w_{i,k,l} \times P_d\left(f_{k,l}\right)\right\}.\end{aligned} \tag{7}$$

Obviously, systems energy consumption $\mathrm{EN}(P)$ is greater than application energy consumption $\mathrm{EN}(V)$. In this paper, one of our main objectives is to minimize systems energy consumption $\mathrm{EN}(P)$.

4. System Reliability Analysis and Problem Statement

In this section, we first provide the single DVFS-enabled processor failure rate model. Then, we analyze heterogeneous systems reliability. At last, we formulate the reliability and energy aware task scheduling as a linear programming problem.

4.1. Single DVFS-Enabled Processor Failure Rate. Among various sources of unreliability in a semiconductor circuit processor, it is predicted that the failure rate due to cosmic ray radiation-induced soft errors dominates all other reliability issues [24]. Transient fault occurs when a high energy particle such as alpha or neutron strikes a sensitive region in a semiconductor device and flips the logical state of the struck node [33]. Most of the modern DVFS-enabled processor is the integration of multibillion transistors on a single chip leading to increasing number of sensitive devices in submicron technologies which is vulnerable to soft error and consequently raises the *Soft Error Rate* (SER) [34]. These phenomena become more and more serious with the continued scaling of processor's voltage and frequency [23, 25].

Traditionally, the modern DVFS-enabled processor's reliability has been modeled as the following Poisson distribution with a failure rate λ when it works at normal voltage and frequency [8, 9, 27, 29, 35]. Moreover, it has been shown that DVFS has a direct and negative effect on failure rates as blindly applying DVFS to scale the supply voltage and processing frequency for energy savings, which may cause significant degradation in processor's reliability [23, 25, 26]. Therefore, for the DVFS-enabled heterogonous processor $p_k \in P$ to be considered in this paper, the failure rate at a reduced frequency $f_{k,l}$ (and the corresponding voltage $V_{k,l}$) can be modeled as

$$\lambda_k(f_{k,l}) = \lambda_k \cdot H_k(f_{k,l}), \tag{8}$$

where λ_k is the failure rate corresponding to the normal processing frequency f_{nm} (and corresponding to normal voltage V_{nm}). Prior researches which studied the effect of normal voltage on processor's reliability have revealed that the failure rates generally increase with scaled processing frequencies (and supply voltages) away from normal voltage [24, 36]. On the other hand, the fault rates are exponentially related to the circuit's critical charge (which is the threshold voltage). Thus, we have the following equations:

$$H_k(f_{k,l}) = \begin{cases} e^{\psi_k V t_k} 10^{\xi_k((f_{k,l}-f_{\mathrm{nm}})/(f_{\max}-f_{\min}))} & f_{\mathrm{nm}} \le f_{k,l} \le f_{\max} \\ e^{\psi_k V t_k} 10^{\xi_k((f_{\mathrm{nm}}-f_{k,l})/(f_{\max}-f_{\min}))} & f_{\min} \le f_{k,l} \le f_{\mathrm{nm}}, \end{cases} \tag{9}$$

where the exponent ψ_k is the parameter of threshold voltage and ξ_k is a constant, representing the sensitivity of fault rates to frequency scaling, and $f_{\min}$ and $f_{\max}$ denote the minimum and maximum frequency, respectively.

In order to get the precision value of parameters ψ_k and ξ_k, we use least squares curve fitting method [37]. Therefore, the natural logarithm of both sides for (9) is

$$\ln(H_k(f_{k,l})) = \begin{cases} \psi_k V t_k + \xi_k \ln 10 \dfrac{f_{k,l} - f_{\mathrm{nm}}}{f_{\max} - f_{\min}} & f_{\mathrm{nm}} \le f_{k,l} \le f_{\max} \\ \psi_k V t_k + \xi_k \ln 10 \dfrac{f_{\mathrm{nm}} - f_{k,l}}{f_{\max} - f_{\min}} & f_{\min} \le f_{k,l} \le f_{\mathrm{nm}}. \end{cases} \tag{10}$$

Let $y = \ln(H_k(f_{k,l}))$, $A = \psi_k V t_k$, $B = \xi_k \ln 10(1/(f_{\max} - f_{\min}))$, and $C = \xi_k \ln 10(f_{\mathrm{nm}}/(f_{\max} - f_{\min}))$. Then, (10) becomes

$$y = \begin{cases} A + B f_{k,l} - C & f_{\mathrm{nm}} \le f_{k,l} \le f_{\max} \\ A - B f_{k,l} + C & f_{\min} \le f_{k,l} \le f_{\mathrm{nm}}. \end{cases} \tag{11}$$

Thus, we can get the parameters ψ_k and ξ_k approximation value by using least squares linear fitting method.

4.2. Application Reliability Analysis. Assume that the task v processing time has taken place during the time interval $[A, B]$ on heterogeneous DVFS-enabled processor p_k at frequency $f_{k,l}$, where A denotes the task start execution time and B denotes the task finish time [5, 8, 9, 29]. Thus, the task execution reliability can be given by

$$\begin{aligned} P[v] &= P\{X(B) - X(A) = 0\} \\ &= P\{X(B - A + A) - X(A) = 0\} \\ &= \exp\{-\lambda_k(f_{k,l})(B - A)\}. \end{aligned} \tag{12}$$

For a task v_i of application G on processor p_k at frequency $f_{k,l}$, its reliability $P[v_i, f_{k,l}]$ is equal to all of its immediate parent tasks and its execution reliability, which can be defined by

$$P[v_i, f_{k,l}] = \prod_{v_j \in \mathrm{pred}(v_i)} P[v_j] \times \exp(-\lambda_k(f_{k,l}) \times w_{i,k,l}), \tag{13}$$

where $\mathrm{pred}(v_i)$ denotes all direct predecessors of v_i and $P[v_j]$ is the reliability of task v_j that is equal to the reliability of task v_i executing on processor p_k at frequency $f_{k,l}$

$$P[v_j] = \sum_{1 \le k \le m}^{1 \le l \le M_k} \{X_{1,k}^l P[v_j, f_{k,l}]\}. \tag{14}$$

For the entry task v_1 of application, which is executed on processor p_k at frequency $f_{k,l}$ and $\mathrm{pred}(v_1) = \phi$, its reliability

$$P[v_1, f_{k,l}] = \exp\left(-\sum_{1 \le k \le m}^{1 \le l \le M_k} \{X_{1,k}^l \lambda_k(f_{k,l}) \times w_{1,k,l}\}\right). \tag{15}$$

Generally, application G has one exit task v_{exit}. The reliability of application $P[G]$ is equal to the exit task v_{exit}:

$$P[G] = P[v_{\mathrm{exit}}] = \prod_{v_j \in \mathrm{pred}(v_{\mathrm{exit}})} P[v_j] \times P[v_{\mathrm{exit}}, f_{k,l}]. \tag{16}$$

This is the other objective of this paper, in which we try to improve the application reliability $P[G]$. From the above analysis, we know that allocating tasks with less execution times to more reliable processors might be a good heuristic to increase the reliability.

Input: The task DAG of parallel applications
Output: The scheduling of task-processor pairs
(1) Calculate each task b_level of DAG
(2) Sort tasks in a scheduling list by non-increasing order of b_level
(3) **while** *the scheduling list isnot empty* **do**
(4) Remove the first task v_i from the scheduling list
(5) Set min $F(v_i)$, min $E(v_i)$ as maximum value
(6) **for** *each processor-frequency* $f_{k,l}$ *in systems* **do**
(7) Compute the earliest finish time EFT(v_i, $f_{k,l}$) use (22)
(8) **if** min $F(v_i)$ > EFT(v_i, $f_{k,l}$) **then**
(9) min $F(v_i)$ = EFT(v_i, $f_{k,l}$)
(10) **end**
(11) Compute task energy consumption EN(v_i, $f_{k,l}$) use (5)
(12) **if** min $E(v_i)$ > EN(v_i, $f_{k,l}$) **then**
(13) min $E(v_i)$ = EN(v_i, $f_{k,l}$)
(14) **end**
(15) **end**
(16) Set min RE(v_i) as maximum value
(17) **for** *each processor-frequency* $f_{k,l}$ *in systems* **do**
(18) Compute the earliest finish time EFT(v_i, $f_{k,l}$) use (22)
(19) Compute task energy consumption EN(v_i, $f_{k,l}$) use (5)
(20) Compute metric RE(v_i, $f_{k,l}$) use (24)
(21) **if** min RE(v_i) > RE(v_i, $f_{k,l}$) **then**
(22) min RE(v_i) = RE(v_i, $f_{k,l}$)
(23) **end**
(24) **end**
(25) Assign task v_i to the corresponding processor-frequency
(26) Update the processor execution finish time
(27) **end**
(28) "Slack reclamation
(29) **for** *each task in scheduling task-processor pairs* **do**
(30) Compute task slack time Slack(v_i) use (25)
(31) **for** *each frequency of processor* k **do**
(32) Compute the optimal frequency $f_{k,l}$ use (26)
(33) **end**
(34) Reassign task v_i and update corresponding data
(35) **end**
(36) Compute the schedule length, application reliability $P[G]$, systems energy consumption EN(P)

Algorithm 1: The pseudocode for REAS algorithm.

4.3. Problem Statement. As simultaneous management of scheduling performance, system reliability, and energy consumption is the main problem of this paper, we formulate it as follows:

$$\begin{aligned} &\text{Minimize} && \text{makespan} \\ &\text{Minimize} && \text{EN}(P) \\ &\text{Maximize} && P[G] \\ &\text{s.t.} && X_{i,k}^{l} = 1 \text{ Or } X_{i,k}^{l} = 0 \\ & && \sum_{1\le k\le m}^{1\le l\le M_k} X_{i,k}^{l} = 1 \quad \forall v_i \in V \\ & && v_i R v_j \quad \forall v_i, v_j \in V. \end{aligned} \tag{17}$$

5. Proposed Reliability-Energy Aware Scheduling Algorithm

This section presents a Reliability-Energy Aware Scheduling algorithm on heterogeneous systems called REAS, which aims at achieving lower energy consumption, high reliability, and shorter schedule length. Its scheduling decisions are made using the hybrid metric including energy consumption, reliability, and schedule length, devised as a novel objective function. The pseudocode of the algorithm is shown in Algorithm 1. The algorithm is complete in three main phases as described in the following sections.

5.1. Task Priorities Phase. This step is essential for list scheduling algorithms. A task processing list is generated by sorting the task by decreasing order of some predefined rank

Table 4: The b_level value of task.

Task	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
b_level	73.4	61.4	42.4	26	34.15	16.6	13.7	3.8
Seq	1	2	3	5	4	6	7	8

function, such as t_level, b_level, Rank, CP, and DL [5, 6, 8–10, 29]. Here, we use the average computation capacity, which is defined as

$$\overline{w(v_i)} = \frac{\sum_{1\le k\le m}^{1\le l\le M_k} w_{i,k,l}}{\sum_{1\le k\le m} M_k}. \tag{18}$$

In this research, we use b_level as the rank function. The b_level of task v_i is the sum of the path weight from task v_i to exit task. We can compute this value recursively traversing DAG from exit task, and it is defined as follows:

$$b\_\text{level}(v_i) = \overline{w(v_i)} + \max_{v_j\in\text{succ}(v_i)}\left\{d_{i,j} + b\_\text{level}(v_j)\right\} + \text{RC}(v_i), \tag{19}$$

where $\text{succ}(v_i)$ is the set of immediate successors of task v_i. $\text{RC}(v_i)$ is the average reliability overhead of task v_i and can be computed by

$$\text{RC}(v_i) = \left(1 - \exp\left\{-\frac{\sum_{1\le k\le m}^{1\le l\le M_k}\lambda_k(f_{k,l})}{\sum_{1\le k\le m} M_k}\times\overline{w(v_i)}\right\}\right)\times\overline{w(v_i)}. \tag{20}$$

For the exit task v_{exit}, the b_level is equal to

$$b\_\text{level}(v_{\text{exit}}) = \overline{w(v_{\text{exit}})} + \text{RC}(v_{\text{exit}}). \tag{21}$$

Basically, $b\_\text{level}(v_i)$ is the length of the critical path from task v_i to the exit task, including the average computation cost and reliability overhead of task v_i. For example, considering the application DAG in Figure 1(b), heterogeneous systems parameters in Table 1, task execution time matrix in Table 2, and communication matrix in Table 3, the task b_level value which is recursively computed by (19) and (21) is shown in Table 4.

5.2. Task Assignment Phase. In this phase, tasks are assigned to the processors with earliest execution finish time $\text{EFT}(v_i)$, high reliability, and minimum task energy consumption $\text{EN}(v_i)$. However, for heterogeneous systems, these performance metrics are conflicted most of the time. Here, we introduce a novel objective as RE, which can get good tradeoff among these metrics. We first redefine task v_i earliest execution finish time on processor p_k at frequency $f_{k,l}$ as

$$\text{EFT}(v_i, f_{k,l}) = \text{EST}(v_i, f_{k,l}) + w_{i,k,l} + \text{RO}(v_i, f_{k,l}), \tag{22}$$

where $\text{RO}(v_i, f_{k,l})$ is the reliability overhead of task v_i on processor p_k at frequency $f_{k,l}$ and is computed by

$$\text{RO}(v_i, f_{k,l}) = (1 - P[v_i, f_{k,l}])\times w_{i,k,l}. \tag{23}$$

On the other hand, we let $\text{Min}\,F(v_i)$, $\text{Min}\,E(v_i)$ denote the earliest execution finish time and minimum task energy consumption on all processors of heterogeneous systems. Thus, the novel metric RE of task v_i on processor p_k at frequency $f_{k,l}$ is

$$\text{RE}(v_i, f_{k,l}) = \theta\times\frac{\text{EFT}(v_i, f_{k,l}) - \text{Min}\,F(v_i)}{\text{EFT}(v_i, f_{k,l})} + (1-\theta)\times\frac{\text{EN}(v_i, f_{k,l}) - \text{Min}\,E(v_i)}{\text{EN}(v_i, f_{k,l})}, \tag{24}$$

where θ is the weight of task earliest execution finish time. If the task execution time is more important than energy consumption, we can give higher value to θ; otherwise, θ value is lower. Moreover, the scheduling objective of this problem is minimum in both schedule length and energy consumption. Thus, in each task assignment step, we try to get the minimum $\text{RE}(v_i, f_{k,l})$ and assign task v_i to the corresponding processor frequency.

5.3. Slack Reclamation. Tasks of parallel application may have some slack time for their execution due primarily to communication events, for example, "multidimensional" intertask communication (or intertask data dependencies), and these processor slacks are an obvious source of energy wastage. Slack reclamation was studied to reduce energy consumption using the slack left by some completed task instances. The idea behind the slack reclamation for the reducing of energy consumption is to exploit the slack time to slow down the execution speeds of the remaining tasks [12, 20]. In this paper, we adopt this technique to reduce energy consumption after making the scheduling decision. The slack time of task v_i is defined by

$$\text{Slack}(v_i) = \min_{v_j\in\text{succ}(v_i)}\begin{cases}\text{Sch}(v_j, sT) - \text{Sch}(v_i, fT) & \text{if } v_i, v_j \text{ on same processor}\\ \text{Sch}(v_j, sT) - d_{i,j} - \text{Sch}(v_i, fT) & \text{otherwise,}\end{cases} \tag{25}$$

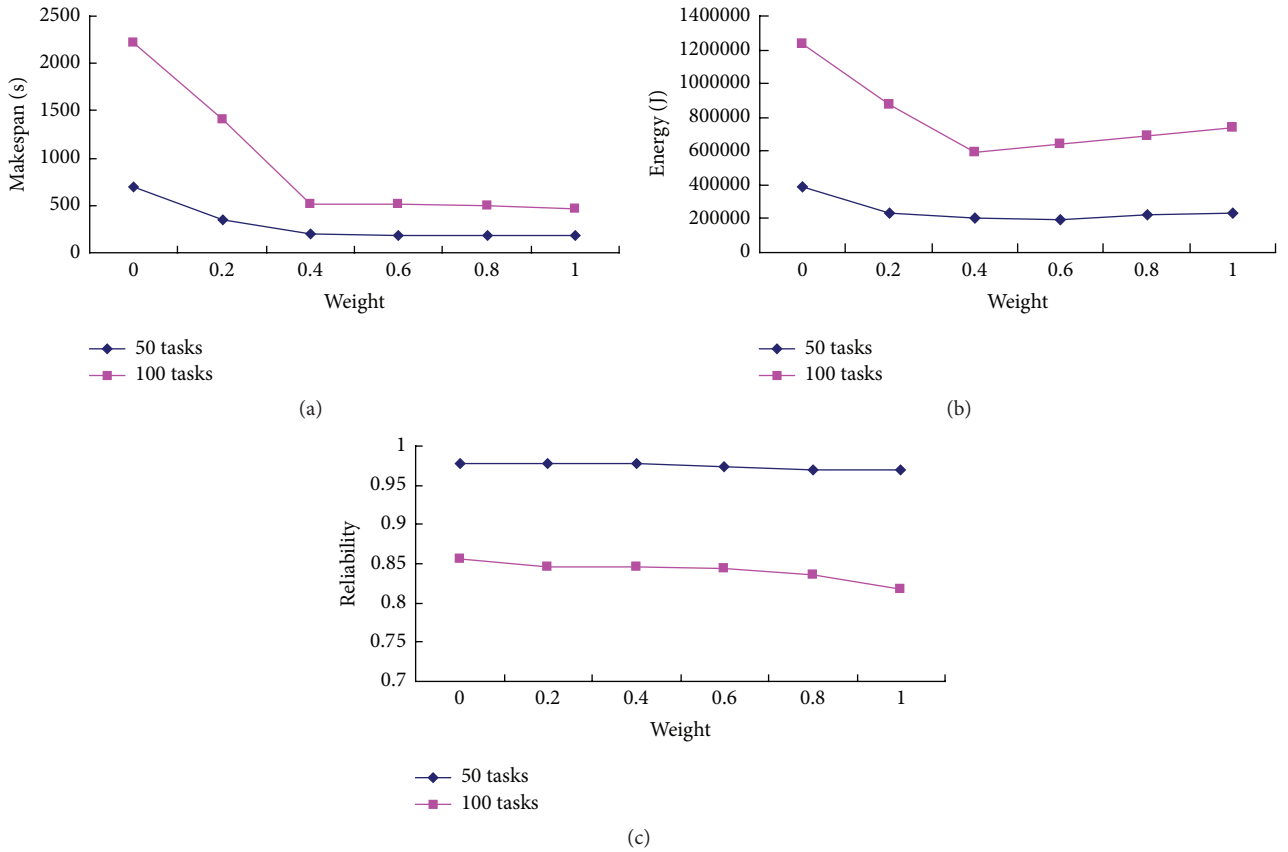


FIGURE 2: The experimental results of REAS algorithm with various weights θ. (a) Schedule length. (b) Energy consumption. (c) Application reliability.

where $\mathrm{Sch}(v_i, sT)$ is the task v_i earliest start time in scheduling processor-frequency pairs and $\mathrm{Sch}(v_i, fT)$ is the earliest finish time.

If task slack time $\mathrm{Slack}(v_i) > 0$, we can scale down the execution frequency to save energy consumption. Thus, the optimal frequency $f_{k,l}$ is satisfied with

$$w_{i,k,l} + \mathrm{RO}(v_i, f_{k,l}) < \mathrm{Sch}(v_i, fT) + \mathrm{Slack}(v_i),$$
$$\mathrm{EN}(v_i, f_{k,l}) < \mathrm{EN}(v_i, f_{k,\mathrm{orig}}), \quad (26)$$

where $f_{k,\mathrm{orig}}$ is the original scheduling processor-frequency pairs. At last step, we reassign task v_i to the optimal frequency $f_{k,l}$.

6. Experimental Results and Discussion

In this section, we compare the performance, energy consumption, and system reliability using our REAS algorithm with three existing scheduling algorithms: DLS [6], RDLS [27], and ECS [20]. The experiments are performed on the synthetic randomly generated precedence-constrained parallel application graphs as described below. The performance metrics chosen for the comparison are the schedule length (2) and (22), systems energy consumption EN(P) (7), and application reliability $P[G]$ (16).

To test the performance of these algorithms, we have developed a discrete event simulation environment of heterogeneous systems with 8 DVFS-enabled processors using C++. This simulator includes 2 Intel® Core™ Duo, 2 Intel Xeon, 2 AMD Athlon, 1 TI DSP, and 1 Tesla GPU, mostly based on Intel processor. The systems are interconnected by Infiniband, which is a switched fabric communications link primarily used in high-performance computing. For the Infiniband configuration, the switch considered is Mellanox InfiniScaleTM III SDR and NIC is Mellanox ConnectXTM IB Dual Copper Card [21]. Other parameters of the model are set as follows. The failure rates of processors are assumed to be uniformly distributed between 1×10^{-4} and 1×10^{-5} failures/hr [8, 9, 28]; the transmission rates of links are assumed to be 1000 Mbits/sec.

6.1. Randomly Generated Application Graphs. These experiments use three commonly DAG characteristics to generate parallel application graphs [5, 8, 9, 29]:

(i) *DAG Size (v).* It is the number of tasks in the application DAG.

(ii) *Communication-Computation Ratio (CCR).* It is the ratio of communication time to computation time. A small CCR value means the application is computation-intensive; a large CCR value indicates

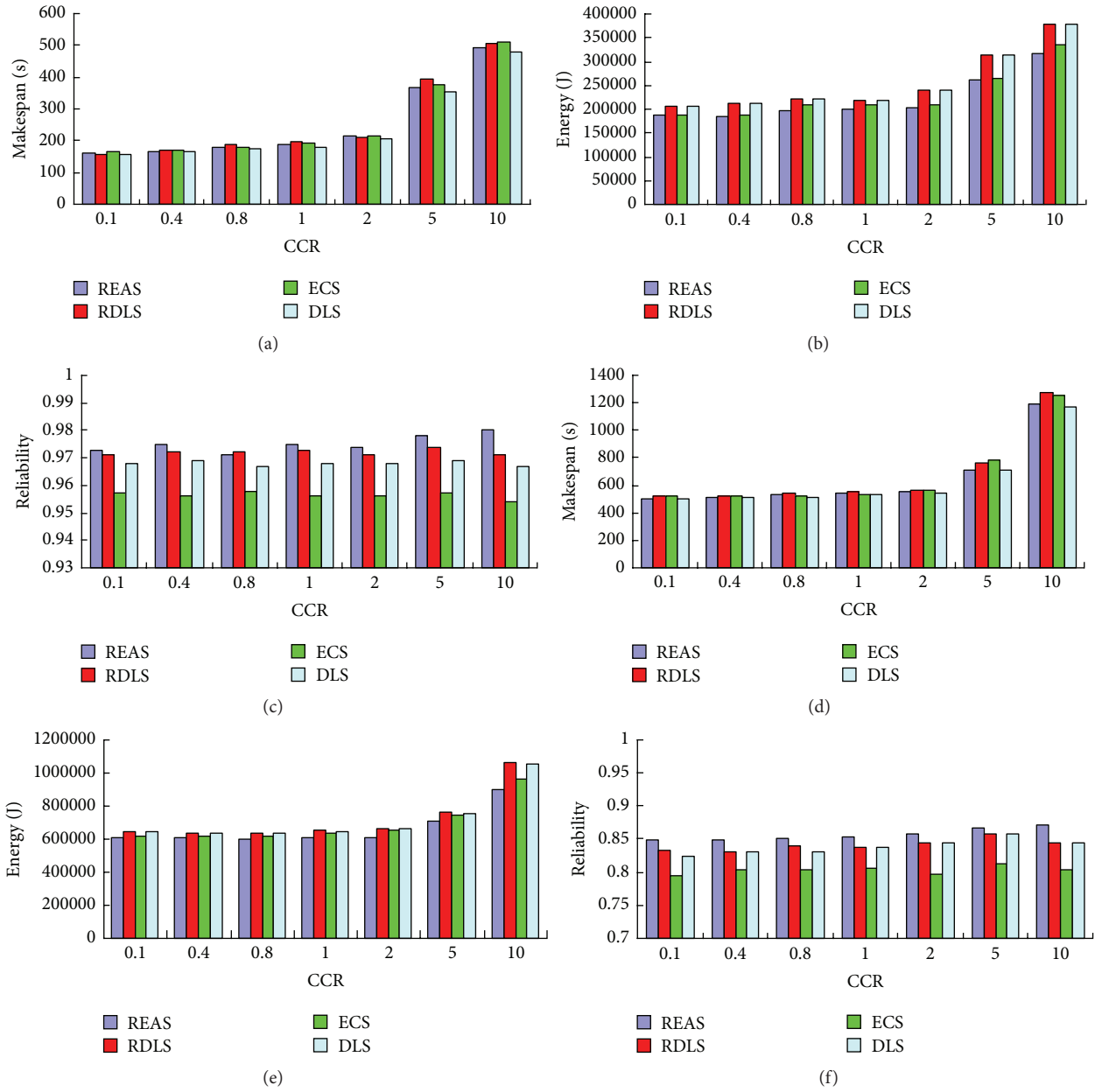


FIGURE 3: The experimental results. (a) 50-task schedule length. (b) 50-task energy consumption. (c) 50-task application reliability. (d) 100-task schedule length. (e) 100-task energy consumption. (f) 100-task application reliability.

that the application is communication-intensive [5, 8–10, 29].

(iii) *Out-Degree*. It is out-degree of a task node.

In experiments setting, DAG are generated based on the above parameters with the number of tasks 50 and 100. Task weights are generated randomly from uniform distribution $[1 \times 10^9, 9 \times 10^{11}]$ execution cycles to be around 4.5×10^{10}; thus the average task execution cycles are 4.5×10^{10}. We also generated edge weights with a uniform distribution based on a mean CCR. Different objective parallel applications can be produced as giving various CCR values [5, 8–10, 29]. In these experiments, we varied CCR in a reasonable range of 0.1 to 10.

6.2. Various Weight θ of REAS Algorithm. In the first experiments, we evaluate the performance of weight θ to REAS algorithm. Figure 2 shows the simulation results of scheduling 50 and 100 tasks with CCR = 1 by varying weight θ from 0 to 1, in steps of 0.2. We observe from Figure 2 that the schedule length and energy consumption decrease and the application reliability almost at the same level as the REAS algorithm weight θ increases. It is reasonable that the REAS algorithm with high θ is mostly based on task execution time and makes its schedule length shorter and consumes less

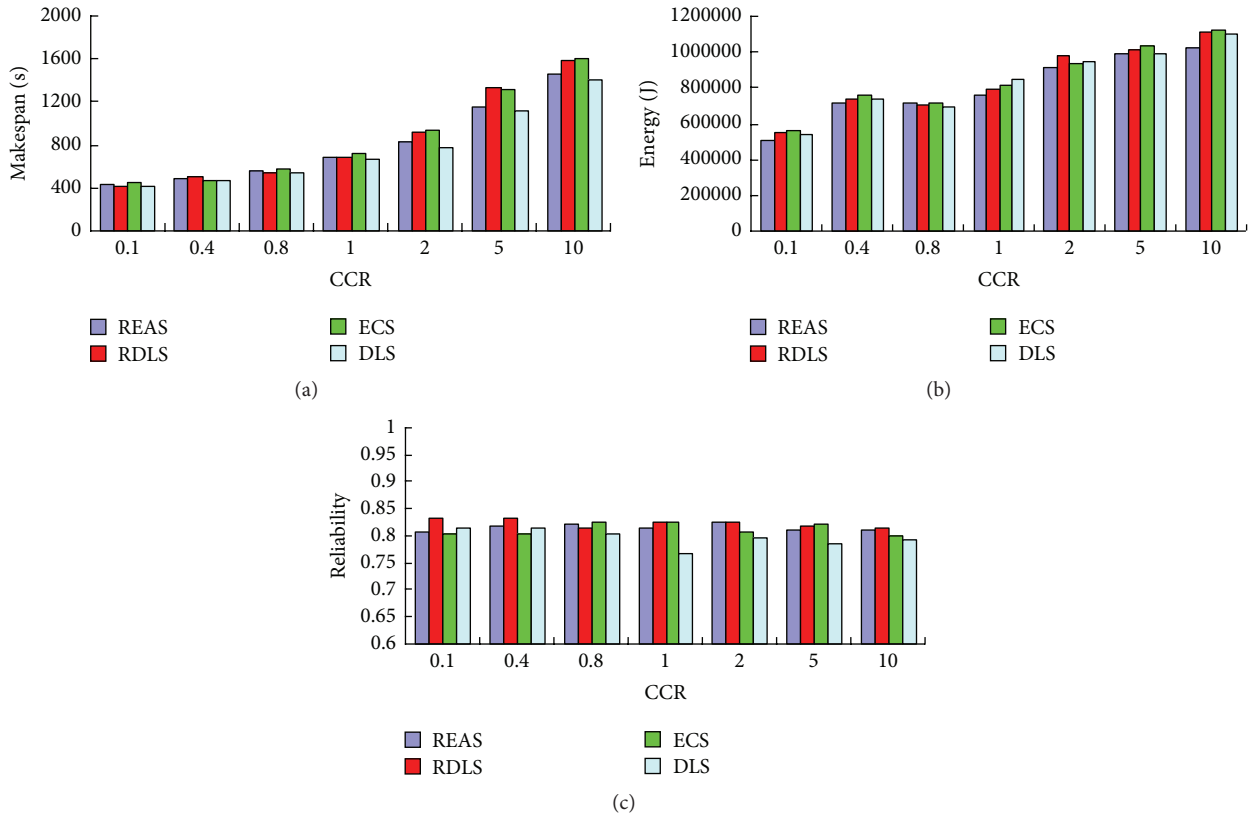


Figure 4: The experimental results of 100 tasks for 4 Intel Xeon and 4 AMD Athlon. (a) Schedule length. (b) Energy consumption. (c) Application reliability.

energy. However, as the weight θ over 0.4, the performance of REAS is not much distinguishable. Thus, in the below experiments, we let $\theta = 0.5$.

6.3. Random Task Performance Results. For the set of randomly generated parallel applications, the results are shown in Figures 3 and 4, where each data point is the average of the data obtained in 1,000 experiments. In this set of experiments, we assume the weight $\theta = 0.5$ of metric RE (see (24)) in REAS algorithm. In other word, the REAS algorithm has the same weight on task execution time and energy consumption. In the next section, we will examine the performance by various weights θ.

We observe from Figure 3(a) that REAS is over RDLS and ECS with respect to schedule length, and the schedule length increases as the CCR increases. The average schedule length of the REAS algorithm is shorter than that of the RDLS and ECS by 2.6% and 1.9%, respectively. This improvement becomes more obvious as CCR increases, for CCR = 5 and REAS over RDLS and ECS by 7.5% and 2.6%, respectively. However, the REAS is inferior to DLS in terms of schedule length. Figure 3(b) reveals that REAS saves more average energy consumption than RDLS by 15.3%, ECS by 3.7%, and DLS by 16%, respectively. Figure 3(c) shows that REAS outperforms RDLS, ECS, and DLS by 0.3%, 2%, and 0.7% in terms of the average application reliability.

This is mainly due to the fact that REAS algorithm schedules tasks according to the novel objective RE, which can get effective tradeoff among task execution time, energy consumption, and task execution reliability. However, DLS algorithm only focuses on optimizing the task execution time and its actual execution time including the task scheduling time and reliability overhead. Thus, the scheduling solution generated by DLS can get optimal schedule length. However, it consumes more energy and has lower reliability. RDLS algorithm schedules tasks considering their execution reliability and ignoring task energy consumption. ECS algorithm is a solution for optimizing both schedule length and energy consumption, but this solution needs more task execution reliability overhead. Thus, REAS algorithm outperforms RDLS, ECS, and DLS in terms of the schedule length, energy consumption, and reliability. Other interesting experimental phenomena are that RDLS and DLS are better than ECS in terms of reliability. This is mainly due to the fact that tasks of solutions RDLS and DLS are always executing on the normal frequency of processor, which has the high reliability in all processor frequency.

The improvements of scheduling performance also could be concluded from Figures 3(d), 3(e), and 3(f) for 100 tasks. These results also show REAS over RDLS and ECS by 4.9% and 3.5% in terms of the average schedule length. And, REAS is also over RDLS, ECS, and DLS by 8.93%, 4.53%, and 8.24% in terms of the average energy consumption and by 1.86%,

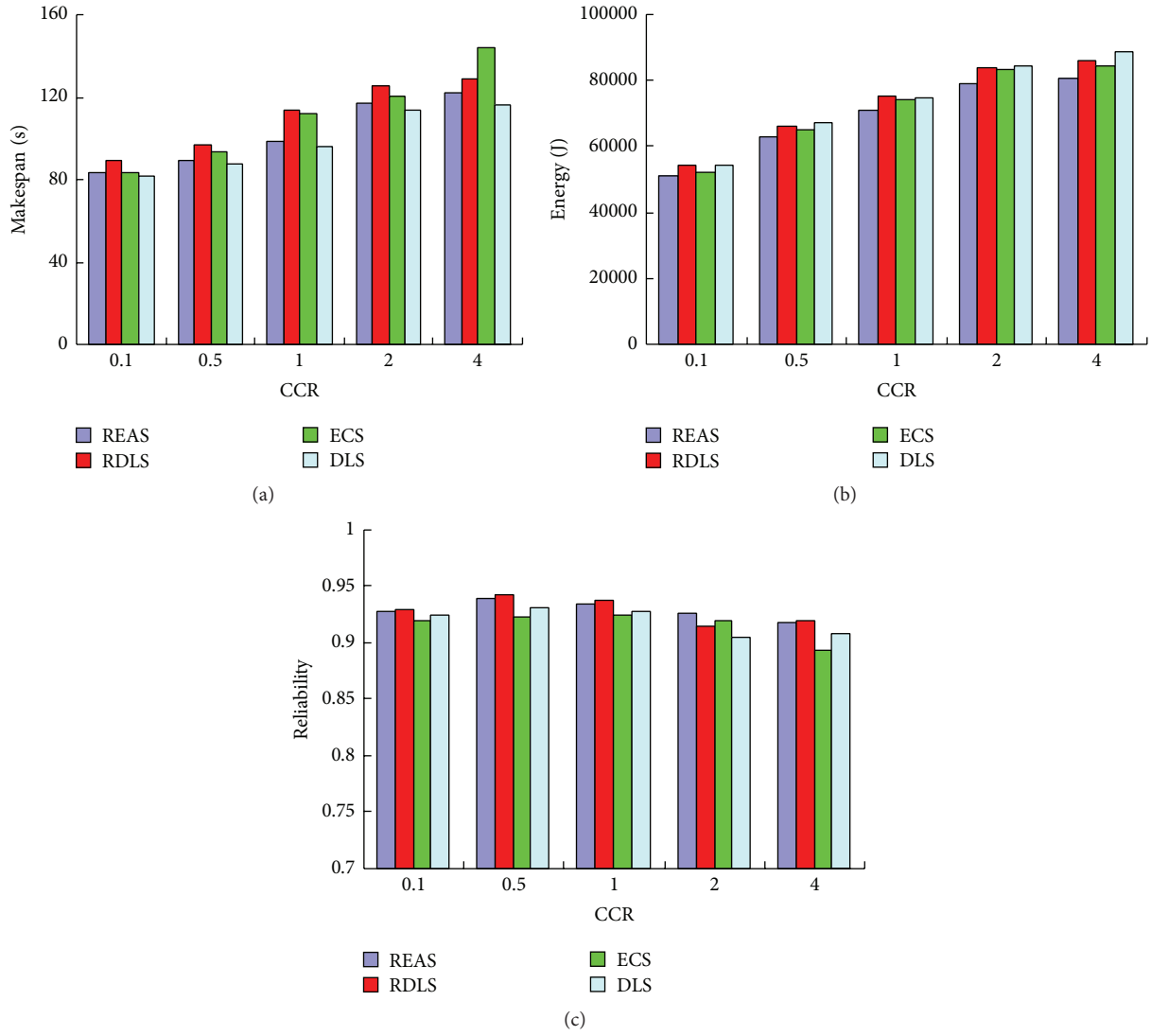


Figure 5: The experimental results of real-world DSP problem. (a) Schedule length. (b) Energy consumption. (c) Application reliability.

6.28%, and 2.1% in terms of the average application reliability, respectively.

We also simulate heterogeneous systems with 4 Intel Xeon and 4 AMD Athlon; the other configurations are the same as before. Figure 4 shows the results of 100 randomly generated tasks on this heterogeneous computing platform. The results show REAS over RDLS, ECS, and DLS in terms of average schedule length and energy consumption. However, REAS is inferior to RDLS in terms of the application reliability.

6.4. Application Graphs of Real-World Problem. Using real applications to test the performance of algorithms is very common [5, 8–10, 29]. In this section, we also simulate a real-world digital signal processing (DSP) problem, and the detail can be seen in [5, 8–10, 29]. From Figure 5, we can conclude that REAS is also better than RDLS, ECS, and DLS.

7. Conclusions and Future Work

In the past few years, with the rapid development of heterogeneous systems, the high price of energy, system performance, reliability, and various environmental issues have forced the high-performance computing sector to reconsider some of its old practices with an aim to create more sustainable system. In this paper, we attempt the simultaneous management of system performance, reliability, and energy consumption. To achieve this goal, we first built a reliability and energy aware task scheduling architecture, which mainly includes heterogeneous systems, parallel application DAG model, and energy consumption model. Then, we proposed a relationship between execution reliability and processor's voltage/frequency and deduced its parameters approximation value by least squares curve fitting method. Thirdly, we established parallel application execution reliability model and formulated this reliability and energy aware scheduling problem as a linear programming. Finally, to provide an

optimum solution for this problem, we proposed a heuristic Reliability-Energy Aware Scheduling (REAS) algorithm based on a novel scheduling objective RE, which is synthetic considering the task execution time, energy consumption, and reliability.

The performance of REAS algorithm is evaluated with an extensive set of simulations and compared to three of the best existing scheduling algorithms for heterogeneous systems: the RDLS, ECS, and DLS algorithms. The comparison is also performed on the synthetic randomly generated precedence-constrained parallel application DAG. The simulation experiment results clearly confirm the superior performance of REAS algorithm over the other three, particularly in energy saving.

This work is one of the first attempts to consider the simultaneous management of system performance, reliability, and energy consumption on high-performance computing systems. Future studies in this domain are twofold. Firstly, it will be interesting to extend our model to multidimensional computing resources, such as interconnections, memory access, and I/O activities. Secondly, in this paper, the failures occurring on resources of systems are assumed to follow Poisson process. Other reliability models can also be used in further studies.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This research was partially funded by the National Science Foundation of China (Grant no. 61370098), Hunan Provincial Natural Science Foundation of China (Grant no. 2015JJ2078), National High-Tech R&D Program of China (2015AA015303), Key Technology Research and Development Programs of Guangdong Province (2015B010108006), and a project supported by the Science Foundation for Postdoctorate Research from the Ministry of Science and Technology of China (Grant no. 2014M552134).

References

[1] http://www.datacenterdynamics.com/focus/archive/2014/01/dcd-industry-census-2013-data-center-power.

[2] http://www.top500.org/lists/2015/11/.

[3] S. Rusu, S. Tam, H. Muljono et al., "A 45 nm 8-core enterprise xeonr processor," *IEEE Journal of Solid-State Circurits*, vol. 45, no. 1, pp. 7–14, 2010.

[4] L. Charng-Da, *Scalable diskless checkpointing for large parallel systems [Ph.D. thesis]*, University of Illinois at Urbana-Champaign, 2005.

[5] X. Tang, K. Li, Z. Zeng, and B. Veeravalli, "A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems," *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 1017–1029, 2011.

[6] F. Dong and G. Selim, "Scheduling algorithms for grid computing: state of the art and open problems," Tech. Rep. 2006-504, 2006.

[7] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Information Sciences*, vol. 270, pp. 255–287, 2014.

[8] X. Tang, K. Li, R. Li, and B. Veeravalli, "Reliability-aware scheduling strategy for heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 70, no. 9, pp. 941–952, 2010.

[9] X. Tang, K. Li, and G. Liao, "An effective reliability-driven technique of allocating tasks on heterogeneous cluster systems," *Cluster Computing*, vol. 17, no. 4, pp. 1413–1425, 2014.

[10] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[11] Y. Xu, K. Li, L. He, and T. K. Truong, "A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization," *Journal of Parallel and Distributed Computing*, vol. 73, no. 9, pp. 1306–1322, 2013.

[12] Y. Wang, K. Li, H. Chen, L. He, and K. Li, "Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, pp. 134–148, 2014.

[13] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys*, vol. 37, no. 3, pp. 195–237, 2005.

[14] K. Li, X. Tang, and Q. Yin, "Energy-aware scheduling algorithm for task execution cycles with normal distribution on heterogeneous computing systems," in *Proceedings of the 41st International Conference on Parallel Processing (ICPP '12)*, pp. 40–47, Pittsburgh, Pa, USA, September 2012.

[15] Z. Du, H. Sun, Y. He, Y. He, D. A. Bader, and H. Zhang, "Energy-efficient scheduling for best-effort interactive services to achieve high response quality," in *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS '13)*, pp. 637–648, Boston, Mass, USA, May 2013.

[16] J.-J. Han, M. Lin, D. Zhu, and L. T. Yang, "Contention-aware energy management scheme for NoC-based multicore real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 691–701, 2015.

[17] J. Mei, K. Li, and K. Li, "Energy-aware task scheduling in heterogeneous computing environments," *Cluster Computing*, vol. 17, no. 2, pp. 537–550, 2014.

[18] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale MPI programs," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '07)*, pp. 1–9, Reno, Nev, USA, November 2007.

[19] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in DVFS-based energy consumption minimization," *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, pp. 1154–1164, 2011.

[20] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374–1381, 2011.

[21] Z. Zong, A. Manzanares, X. Ruan, and X. Qin, "EAD and PEBD: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters," *IEEE Transactions on Computers*, vol. 60, no. 3, pp. 360–374, 2011.

[22] E. Chielle, F. Lima Kastensmidt, and S. Cuenca-Asensi, "Tuning software-based fault-tolerance techniques for power optimization," in *Proceedings of the 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS '14)*, pp. 1–7, Palma de Mallorca, Spain, October 2014.

[23] D. Ernst, S. Das, S. Lee et al., "Razor: circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, 2004.

[24] F. Firouzi, A. Yazdanbakhsh, H. Dorosti, and S. M. Fakhraie, "Dynamic soft error hardening via joint body biasing and dynamic voltage scaling," in *Proceedings of the 14th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD '11)*, pp. 385–392, Oulu, Finland, March 2011.

[25] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 35–40, IEEE, November 2004.

[26] D. Zhu and H. Aydin, "Reliability-aware energy management for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 58, no. 10, pp. 1382–1397, 2009.

[27] A. Dogan and F. Özgüner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 308–323, 2002.

[28] X. Qin and H. Jiang, "A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters," *Journal of Parallel and Distributed Computing*, vol. 65, no. 8, pp. 885–900, 2005.

[29] Y. Xu, K. Li, L. He, L. Zhang, and K. Li, "A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems," *IEEE Transaction on Parallel and Distributed System*, vol. 26, no. 12, pp. 3208–3222, 2015.

[30] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green governors: a framework for continuously adaptive DVFS," in *Proceedings of the International Green Computing Conference (IGCC '11)*, pp. 1–8, IEEE, Orlando, Fla, USA, July 2011.

[31] M. Qiu and E. H.-M. Sha, "Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, article 25, 2009.

[32] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of HPC applications on distributed Cloud-oriented data centers," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 732–749, 2011.

[33] R. Baumann, "The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction," in *Proceedings of the International Electron Devices Meeting (IEDM '02)*, pp. 329–332, San Francisco, Calif, USA, December 2002.

[34] A. M. Fard, M. Ghasemi, and M. Kargahi, "Response-time minimization in soft real-time systems with temperature-affected reliability constraint," in *Proceedings of the CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST '15)*, Tehran, Iran, October 2015.

[35] S. Song, D. W. Coit, Q. Feng, and H. Peng, "Reliability analysis for multi-component systems subject to multiple dependent competing failure processes," *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 331–345, 2014.

[36] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin, "Soft errors issues in low-power caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 10, pp. 1157–1166, 2005.

[37] Z. Lei, G. Tianqi, Z. Ji, J. Shijun, S. Qingzhou, and H. Ming, "An adaptive moving total least squares method for curve fitting," *Measurement*, vol. 49, pp. 107–112, 2014.