HPC Software and Programming Environments for Big Data Applications

Lead Guest Editor: Youngjae Kim
Guest Editors: Sungyong Park, Feiyi Wang, Guangyu Sun, and Shangguang Wang
HPC Software and Programming Environments for Big Data Applications
HPC Software and Programming Environments for Big Data Applications

Lead Guest Editor: Youngjae Kim
Guest Editors: Sungyong Park, Feiyi Wang, Guangyu Sun, and Shangguang Wang
Editorial Board

Marco Aldinucci, Italy  
Davide Ancona, Italy  
Ferruccio Damiani, Italy  
Sergio Di Martino, Italy  
Basilio B. Fraguela, Spain  
Carmine Gravino, Italy  
Gianluigi Greco, Italy  
Bormin Huang, USA  
Chin-Yu Huang, Taiwan  
Jorn W. Janneck, Sweden  

Christoph Kessler, Sweden  
Harald Köstler, Germany  
José E. Labra, Spain  
Thomas Leich, Germany  
Piotr Luszczek, USA  
Tomàs Margalef, Spain  
Cristian Mateos, Argentina  
Roberto Natella, Italy  
Francisco Ortin, Spain  
Can Özturan, Turkey  

Antonio J. Peña, Spain  
Danilo Pianini, Italy  
Fabrizio Riguzzi, Italy  
Michele Risi, Italy  
Damian Rouson, USA  
Giuseppe Scanniello, Italy  
Ahmet Soylu, Norway  
Emiliano Tramontana, Italy  
Autilia Vitiello, Italy  
Jan Weglarz, Poland
Contents

HPC Software and Programming Environments for Big Data Applications
Youngjae Kim, Sungyong Park, Feiyi Wang, Guangyu Sun, and Shangguang Wang

Resource Scheduling Based on Improved Spectral Clustering Algorithm in Edge Computing
Guangshun Li, Shuzhen Xu, Junhua Wu, and Heng Ding
Research Article (13 pages), Article ID 6860359, Volume 2018 (2018)

SEDC-Based Hardware-Level Fault Tolerance and Fault Secure Checker Design for Big Data and Cloud Computing
Zahid Ali Siddiqui, Jeong-A Lee, and Unsang Park
Research Article (16 pages), Article ID 7306837, Volume 2018 (2018)

Research on the Prewarning Method for the Safety of South-to-North Water Transfer Project Driven by Monitoring Data
Yang Liu, Yaoling Fan, Xinqing Yan, Xuemei Liu, and Bin Yang
Research Article (7 pages), Article ID 3287065, Volume 2018 (2018)

Big Data in Cloud Computing: A Resource Management Perspective
Saeed Ullah, M. Daud Awan, and M. Sikander Hayat Khiyal
Review Article (17 pages), Article ID 5418679, Volume 2018 (2018)

NUMA-Aware Thread Scheduling for Big Data Transfers over Terabits Network Infrastructure
Taeuk Kim, Awais Khan, Youngjae Kim, Preethika Kasu, and Scott Atchley
Research Article (8 pages), Article ID 4120561, Volume 2018 (2018)

Deployment Strategy for Car-Sharing Depots by Clustering Urban Traffic Big Data Based on Affinity Propagation
Zhihan Liu, Yi Jia, and Xiaolu Zhu
Research Article (9 pages), Article ID 3907513, Volume 2018 (2018)

Mitigating Interference between Scientific Applications in OS-Level Virtualized Environments
Theodora Adufu and Yoonhee Kim
Research Article (9 pages), Article ID 7548527, Volume 2018 (2018)

Cultural Distance-Aware Service Recommendation Approach in Mobile Edge Computing
Yan Li and Yan Guo
Research Article (8 pages), Article ID 2181974, Volume 2018 (2018)

Nonmetric Correction of Lens Distortion Based on Entropy Measure
Tianfei Chen, Lijun Sun, Qiuwen Zhang, Xiang Wu, and Defeng Wu
Research Article (9 pages), Article ID 4230185, Volume 2018 (2018)

Field Geometric Calibration Method for Line Structured Light Sensor Using Single Circular Target
Tianfei Chen, Lijun Sun, Qiuwen Zhang, Xiang Wu, and Defeng Wu
Research Article (8 pages), Article ID 1526706, Volume 2017 (2018)
Editorial

HPC Software and Programming Environments for Big Data Applications

Youngjae Kim, Sungyong Park, Feiyi Wang, Guangyu Sun, and Shangguang Wang

1 Sogang University, Seoul, Republic of Korea
2 Oak Ridge National Laboratory, Oak Ridge, TN, USA
3 Peking University, Beijing, China
4 Beijing University of Posts and Telecommunications, Beijing, China

Correspondence should be addressed to Youngjae Kim; youkim@sogang.ac.kr

Received 26 June 2018; Accepted 17 July 2018; Published 2 September 2018

The complexity of scientific and engineering problems has increased significantly as data have been generated at high speed. Traditional HPC systems have begun to provide a large data-processing infrastructure for high-performance big data processing. In particular, HPC software and programming environments provide a rich computing framework for large data issues that have not previously been resolved. However, traditional HPC system software and programming models were not originally designed and developed for big data application processing and were difficult to use. This special edition includes articles that address issues that improve existing software and programming environment and use them for high-performance big data processing.

This special issue includes nine papers. It covers hardware and software issues ranging from a paper to improve the performance of the big data framework to a paper to improve the performance of the application using the big data framework. The paper “EDC-Based Hardware-Level Fault Tolerance and Fault Secure Checker Design for Big Data and Cloud Computing” describes the hardware-level security concern for big data systems. The paper “Research on the Prewarning Method for the Safety of South-to-North Water Transfer Project Driven by Monitoring Data” proposes an intelligent prewarning method for the abnormal data processing in the South-to-North Water Transfer Project. The paper “Big Data in Cloud Computing: A Resource Management Perspective” identifies key features that characterize big data resource management systems. The paper “NUMA-Aware Thread Scheduling for Big Data Transfers over Terabits Network Infrastructure” proposes a NUMA (Non-Uniform Memory Access)-aware thread and resource scheduling for optimized big data transfers in terabit network. The paper “Deployment Strategy for Car-Sharing Depots by Clustering Urban Traffic Big Data Based on Affinity Propagation” presents an optimization method to determine the depot locations by clustering taxi OD points with the AP (Affinity Propagation) clustering algorithm. The paper “Mitigating Interference between Scientific Applications in OS-Level Virtualized Environments” describes an interference-aware scheduling method that mitigates the problem of performance interference based on applications’ I/O and CPU usage profiles in OS-level virtualized environments. The paper “Cultural Distance-Aware Service Recommendation Approach in Mobile Edge Computing” proposes a cultural distance-aware service recommendation approach in mobile edge computing. The paper “Nonmetric Correction of Lens Distortion Based on Entropy Measure” presents a nonmetric correction algorithm for lens distortion based on entropy measure for the real vision system. The paper “Field Geometric Calibration Method for Line Structured Light Sensor Using Single Circular Target” presents a geometric calibration method for line structured light sensor.

Acknowledgments

We would like to thank the authors and the reviewers for making this special issue possible. Youngjae Kim would like to acknowledge the support from Institute for Information
& communication Technology (IITP) grant funded by the Korea Government (MSIT) (no. 2015-0-00590, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development).

Youngjae Kim
Sungyong Park
Feiyi Wang
Guangyu Sun
Shangguang Wang
Research Article

Resource Scheduling Based on Improved Spectral Clustering Algorithm in Edge Computing

Guangshun Li, Shuzhen Xu, Junhua Wu, and Heng Ding

School of Information Science and Engineering, Qufu Normal University, Rizhao 276800, China

Correspondence should be addressed to Guangshun Li; 30752585@qq.com

Received 27 October 2017; Revised 28 March 2018; Accepted 17 April 2018; Published 8 July 2018

Academic Editor: Feiyi Wang

Copyright © 2018 Guangshun Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the development of Internet of Things (IoT), the massive data generated by it forms big data, and the complexity of dealing with big data brings challenges to resource scheduling in edge computing. In order to solve the problem of resource scheduling and improve the satisfaction of users in edge computing environment, we propose a user-oriented improved spectral clustering scheduling algorithm (ISCM) in this paper. Based on the improved k-means algorithm, the ISCM algorithm solves the problem that the clustering result is sensitive to the initial value and realizes the reclustering, which makes the obtained clustering results more stable. Finally, the edge computing resource scheduling scheme is obtained based on the clustering results. The experimental results show that the resource scheduling scheme based on improved spectral clustering algorithm is superior to traditional spectral clustering algorithm in edge computing environment.

1. Introduction

Edge computing is an extension of cloud computing; it is a way to quickly process data at the edge of the network [1–3].

The goal of the development of Internet of Things (IoT) is to connect electronic devices, mobile terminals, household appliances, and so on [4–6]. These devices are not only large in number but also widespread in geographical distribution. In the real world, when users use these IoT devices, there are various requirements for cloud computing, such as low latency and location awareness [7–10]. In addition, with the advent of the big data era, traditional data analysis methods and storage technology encounter bottlenecks. Big data has the characteristics of large data volume, diversity of data structure, and the requirements of fast processing [11, 12]. However, the reception and transmission of large amounts of data may cause the I/O bottleneck between the cloud computing data center and the terminal; the transmission rate and the processing speed are greatly reduced and even cause a large resource scheduling delay.

Processing range of edge computing is closer to the data source. After the data collection, it is processed at the edge of the network rather than in the central server; it provides edge intelligent services nearly with low latency and location awareness and other features and support mobility. In the Internet of Things, the edge computing through the data analysis and processing not only realizes the sensing, interaction, and control between objects, but also provides real-time resource scheduling and allocation for users [13]. Obviously, edge computing can meet the needs of these IoT devices.

With the various demand for users [14], users can choose to store data to the local edge computing or send it to the cloud computing. In this case, resource allocation and scheduling are user oriented, so task-oriented scheduling algorithms are not applicable. At present, most of the existing scheduling algorithms are applied to grid computing and virtual machine scheduling; it can not satisfy the characteristics of edge computing, such as low latency, close to the end user, wide geographical distribution, and other characteristics, and the general scheduling algorithm can not meet the diversified demands of users. To solve these problems, this paper proposes a resource scheduling algorithm for end users, which aims at solving the resource scheduling problem in the edge computing.
2. Related Work

As a more mature service platform, cloud computing plays an important role in dealing with big data. In addition, there are many cloud computing resource scheduling and resource allocation algorithms [15–23].

In 2016, HeeSeok Choi et al. [24] proposed a task consolidation algorithm based on task classification and resource utilization for the cloud data center. Furthermore, they designed a VM consolidation algorithm to balance task execution time and energy consumption without violating a predefined service level agreement (SLA).

In 2016, Wu et al. [25] proposed a scheduling scheme for continuous clustering of mobile cloud resources based on the improved FCM (IGAFCM) algorithm to reduce the size of matching requirements in the search process. In addition, the experiment proved that the matching strategy can be dynamically adjusted according to the matching score and feedback training.

In 2016, Cheol-Ho Hong et al. [26] proposed 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.

Edge computing is a new platform that can serve local mobile devices. Edge computing makes resource sharing or resource caching among widely deployed edge computing clusters. Next, we introduce some scholars about the work of edge computing in resource management.

In 2015, Su et al. [27] studied the resource caching scheme in edge computing based on the Steiner tree. When the edge computing server caches resources, the Steiner tree is first generated to minimize the total path weight (cost); this Steiner tree can minimize the cost of resource cache. Then the article gave a running example, and compared with the shortest path scheme, the results show that based on the Steiner tree scheme can work more effectively.

In 2015, Aazam et al. [28] proposed a dynamic resource allocation through the Fog Micro Datacenter, which argued that edge computing is between the Internet of Things and the cloud. Its purpose is to manage resources, to achieve data filtering, data preprocessing and data encryption. For this purpose, the paper proposed a resource management architecture in an edge computing environment. The article took into account the factors that give up the customer, the type of service, the service price, and the different waiver probability fluctuations. The experimental results show that these factors can help the service provider to correctly estimate the number of resources.

In 2015, Datta et al. [29] proposed edge computing and consumer-centric networking, which discussed the edge computing paradigm and how it served the consumer-centric IoT. The edge computing architecture can be integrated into an one M2M standard system.

The resource scheduling algorithm proposed in this paper is based on clustering method. Cluster analysis is a multivariate statistical analysis and a part of unsupervised pattern recognition [30–32]. Compared with k-means clustering algorithm, spectral clustering has significant advantages, it can identify any shape of the cluster, such as nonconvex clusters, and it is difficult to fall into the local optimal solution, so that the clustering algorithm has a certain degree of improvement compared with the traditional clustering algorithm. In 2015, Zhou et al. [33] proposed a spectral clustering algorithm to solve the problem of resource scheduling and allocation in cloud computing. However, it is not difficult to see from the implementation steps of the spectral clustering algorithm, and a k-means algorithm is required in the last step. Because the k-means algorithm is sensitive to the initial value, the spectral clustering algorithm is difficult to ensure the stability of clustering results. In this respect, Duan et al. [34] put forward an improved k-means algorithm, and the improved k-means algorithm uses the "shooting target" principle to search the clustering center.

3. Resource Scheduling Architecture

In the edge computing environment supported by the Internet of Things, there are two forms of service available to the end user: one is based on the amount of task and the size of the load, and the end user submits the task to the edge computing micro data center. The other is that the user will submit the resources to the edge computing micro data center, and the edge computing micro data center allocates the resource according to the request submitted by the end user. There are two forms of service, the former we call task oriented, and the latter we call user oriented. The difference between these two forms of service is as follows. For task-oriented resource scheduling, there is no explicit requirement on the computing capability of resources. According to the submitted the amount of task load and task calculation, the edge computing micro data center allocates resources with corresponding processing capabilities to execute tasks. For user-oriented resource scheduling, the user submits resource allocation requests to the edge computing micro data center. The requests of users include explicit requirements on the computing capability of the resource and a clear resource occupancy time interval. In this paper, we consider the problem of resource scheduling and allocation in the edge computing according to the user-oriented service form.

In the framework of the Internet of Things, edge computing is between cloud computing and Internet of Things equipment [38, 39], the three levels shown in Figure 1. The edge computing preprocesses the collected data and then uploads it to the cloud data center; this can greatly reduce the pressure on the cloud data center.

In edge computing, the architecture of the resource scheduling process is shown in Figure 2, the user submits the request to the edge micro data center (EMDC), and the edge server allocates the resource according to the request of users. If the resource requested by the current user is temporarily short of the edge center (Edge1), the edge server can send broadcast messages to its adjacent edge center (Edge2). Edge1
submits the request to Edge2 to process; if Edge2 does not have the resources that the user wants to request, then Edge2 can upload the user’s request to its neighboring Edge3 to process. Edge servers in edge computing are mostly made up of weak performance computers, and the system resources are limited. When some users submit requests, if it exceeds the capability range of the edge computation processing data, then the edge computing will choose to transfer the data of users to the cloud server. In this paper, we study the resource scheduling problem in edge computing, so we do not consider the resource scheduling process after the data upload to the cloud.

4. Solving Scheduling Problems

4.1. Mathematical Description of Scheduling Problem. The request refers to the resource allocation request that the user submits to the edge computing micro data center. It mainly
includes the following aspects: computing resource capability (usually expressed by millions of instructions per second); time interval (including start time, completion time). In order to simplify the calculation process, the number of computing resources is used to represent the demand of users for computing resource capability.

The resource refers to the resources in the edge computing micro data center, such as computing resources, communication resources, and storage resources. When a user requests resource allocation to the edge computing, if the same computing unit is required by different users, there is an overlap in the use time interval of the resource, resulting in resource contention, reduced resource utilization, and service delay. In this paper, we call this overlap in time interval be overlap degree. The size of the overlap degree depends on the time interval of the user occupied the computing unit.

In this paper, the clustering algorithm solves the problem that reduce the overlap degree of user requests and improving resource utilization. The object of clustering is the requests of users. Clustering is based on the degree of time overlap between requests. The greater the overlap degree is, the higher the similarity is. By clustering, requests with a big degree overlap can be in the same category, and requests with a small degree of overlap can be assigned to different categories. During scheduling, resources can be allocated to different categories of requests at the same time to reduce the possibility of resource contention and improve service quality.

Each request of the user has a clear resource occupancy time interval (including the start time and completion time), and different user requests may overlap on the time interval. The overlap degree between all users constitutes a matrix, represented by $M$.

The matrix $M$ has $n \times n$ elements, each of which can be defined as follows:

$$M_{ij} = \frac{P_{ij}}{T} \quad (1)$$

In formula (1), $P_{ij}$ is the overlap time between user i and user j. The time interval $T$ is delimited by the edge computing, and there are multiple requests arriving at this time interval. As shown in Figure 3, the time interval $T$ is from 0 to 9, for a total of 10 time intervals, $T=10$. The earliest start request is user1, and the latest finish request is user4. The user1 request time interval is [0, 4], and user4 request time interval is [3, 9]. The degree of time overlap exists if and only if two users request the same resource. As shaded by the slash lines, when user1 and user2 request the same resource, there is an overlap between the time intervals of the two users. Therefore, according to formula (1), the overlap degree between the user 1 and the user 2 is $3/10 = 0.3$.

Assume at time 0 edge computing micro data center receives the $N$ service request, the amount of resources available is $m$, and the calculation ability of a single resource is constant, so we believe that there is no difference between the two independent computing units (nodes) on the ability. Different users have different requirements for computing capacity. According to the needs of users, multiple resources can be assigned to a user at the same time.

Let $S = \{s_1, s_2, \ldots, s_n\}$ be the user set that would be executed on computing unit $r$; set $S$ refers to the set of all users. For two different units $r_1$ and $r_2$, a user $S_i$ may apply for the use of two computing units $r_1$ and $r_2$, so the intersection between the sets $S_1$ and $S_2$ is not empty. In fact, because a request can take up multiple computing units, which adds to the complexity of the problem. So, we need to divide the requests that take up multiple units. If the request of user needs to take up $a$ units, it should be divided into $a$ requests that have the same time interval with the previous request. So the request set $S = \{s_1, s_2, \ldots, s_n\}$ can be extended to set $\hat{S} = \{s_1, s_2, \ldots, s_m, s_{m+1}, \ldots, s_{m+A}\}$ where $A = \sum_{i=1}^{n} a(s_i) - 1$ and $a(s_i)$ is the number of units required to represent the $i^{th}$ request.

We suppose that the service requests set $S$ is generated randomly, $S = \{S_1, S_2, \ldots, S_n\}, n=30$. Suppose the 30 requests are divided into 6 groups in turn; the number of units occupied by each group is represented by a set $G = \{G_1, G_2, G_3, G_4, G_5, G_6\}$ where $G_1=3$, $G_2=2$, $G_3=5$, $G_4=1$, $G_5=2$, $G_6=1$. According to the formula $A = \sum_{i=1}^{n} a(s_i) - 1$, so that the request set $S = \{s_1, s_2, \ldots, s_n\}$ can be extended to set $\hat{S} = \{s_1, s_2, \ldots, s_{m}, s_{m+1}, \ldots, s_{m+A}\}$, we get $A = 3 \times 5 + 2 \times 5 + 5 \times 5 + 5 + 2 \times 5 + 5 - 1 = 69$, so the set $\hat{S} = \{S_1, S_2, \ldots, S_{31}, \ldots, S_{99}\}$.

With regard to the process of resource allocation, the number of computing units in operation is $k$, and $k$ has
the minimum value; that is, there is a boundary beyond the marginal calculation of micro data center services. Obviously, this minimum corresponds to the maximum value required by all users for computing power. Let \( k_{\min} \) be the minimum value, and we know that \( k_{\min} = \max\{a_k \mid s \in S\} \) from a previous analysis. For example, if the requirement for computing power is five units, the edge computing micro data center does not assign four or fewer units to the user. At the same time, the number of computational units in the operation also has the maximum value. Obviously, the maximum will not exceed the number of available computing units. Let \( k_{\max} \) be maximum value and \( k_{\max} \leq m \). Obviously, \( k \in [k_{\min}, k_{\max}] \).

### 4.2. Scheduling Objectives

In edge computing, resource utilization and quality of service (QoS) should be taken into account in resource scheduling for end users. These are very important reference factors for resource scheduling [40–43]. Resource utilization affects economic benefit, and QoS affects customer satisfaction.

There are two users service request \( s_i \) and \( s_j \); \( p_{ij} \) means overlap degree. We consider the simplification of standardization and computation, let \( c_{ij} = 1 - \beta(1 - p_{ij}) \), where \( \beta \) is a constant and \( 0 < \beta < 1 \). From this, we can see that the higher the overlap degree is, the greater \( c_{ij} \) is.

Therefore, one of the objectives of scheduling can be expressed in this way, and a strategy satisfying the following formula is found:

\[
\min \sum_{r=1}^{m} \sum_{i \in S_r, j \in S_r} c_{ij} \tag{2}
\]

By formula (2), we can get the most satisfying policy for the user. However, it does not consider the optimization problem of resources, which is also one of the objectives of scheduling. For example, when the user requirements are not demanding on the computing ability, the edge computing micro data center will dispatch a small number of computing units to meet the request of users, and the resource utilization can be improved by reducing the number of computing units.

The length of the known time interval is \( T \), and for each computing unit \( r \), the resource utilization rate can be expressed as

\[
U = \sum_{r=1}^{k} \frac{t_r}{kT} \tag{3}
\]

From the perspective of the user, the user request is including the use time of resources and number of resources, which means that time of \( T \) and the number of each user application resources are fixed. Therefore, the value \( \sum_{r=1}^{k} t_r \) is constant, nothing to do with the value of \( k \). According to formula (3), the size of resource utilization is determined by \( k \). When the demand of users is responded, in order to improve resource utilization, we have to reduce the number of resources in the operation. However, reducing the amount of resources that can be put into operation will inevitably increase the overlap degree. This creates a problem that the system must make a choice between the number of resources and the overlap degree.

As mentioned earlier, overlap affects the quality of service (QoS) and reduces user satisfaction. As we all know, there are many factors that affect the quality of service (QoS). This paper focuses on the resource scheduling and allocation problem. In this paper, QoS is expressed as delay extent, actual finish time compared to the expected completion time of services. The QoS condition is defined as the following representation:

\[
\delta = \frac{E}{T} \tag{4}
\]

In summary, under the premise of satisfying the QoS condition, the goal of our resource scheduling can be expressed as

\[
\min \sum_{r=1}^{m} \sum_{i \in S_r, j \in S_r} c_{ij} \max \sum_{r=1}^{k} \frac{t_r}{kT} \tag{5}
\]

### 4.3. Solutions

As mentioned earlier, our goal is to find a formula to satisfy formula (5). If we regard the overlap degree as Euclidean distance, then large-scale requests should form a cluster structure, and the distance between different clusters is very large; in addition, the overlap degree between different clusters is very small. However, the smaller distance between nodes in the cluster is, the bigger overlap degree between nodes in the cluster is. This means that we can turn the scheduling problem into a clustering problem.

The detailed process of converting a scheduling problem to a clustering problem is as follows.

Firstly, a set \( S \) is set up to store the requests of users, and the edge computing allocates resources required by the user. Secondly, the overlap degree between each two requests in the set \( S \) is calculated according to formula (1). It should be noted that a certain user request may require multiple resources and overlap with other requests for multiple times. In this case, the request needs to be divided into multiple subrequests. The set \( S \) is expanded into a set \( \tilde{S} \), and the Euclidean distance between the requests is calculated to construct a matrix of overlap degree.

Finally, the improved spectral clustering algorithm is used to solve the clustering problem. The overlap degree matrix is regarded as a similarity matrix, and the spectral clustering algorithm reduces the dimensions of the matrix. The improved k-means is used to complete the clustering.

According to the final clustering result, in order to facilitate the search for a better scheduling strategy, the requests located in different clusters are selected, and a set of scheduling strategies needs to be established to store the selected requests. Obviously, the overlap degree between requests within the set of scheduling policies is very small, and the overlap degree between requests from different scheduling sets is large.
4.3.1. Improved k-Means Algorithm. In order to make the clustering result that insensitive to the initial value, in this paper, we use the improved k-means algorithm proposed by Duan et al. and then combined it with the spectral clustering algorithm to achieve a relatively stable clustering results.

The improved k-means algorithm (referred to as mykmeans) mainly adds the initial clustering center selection algorithm. The initial number of cluster centers is k, the number of searches is C, and the set B is used to store C×k cluster centers, B = {b_1, b_2, ..., b_C×k}. With the algorithm loop search clustering center, each time the selection results are put into H_t, the set H stores the final cluster center selection results, H = min(distortion(H_t, B)).

In the clustering algorithm, since the initial clustering center selection algorithm is added. When selecting the optimal initial center of the next cluster, the similarity degree matrix (the distance between all points in the dataset) of the entire data set needs to be calculated at each step, which makes the algorithm cost time is very large or needs to store the calculated similarity matrix back up, which in turn requires a lot of storage space. In short, when the algorithm is applied to a data set with a slightly larger amount of data, there is a problem that the calculation time is too long, or the storage space is insufficient, and it is not suitable for large-scale systems with many nodes.

In order to solve the above problems and reduce the computational complexity of the clustering algorithm, in this section we use the improved clustering algorithm proposed by Zhao [44].

When we select the cluster center of the next cluster, if the selected point is in close proximity to the found cluster center, then there will be very little difference in the center values of the two clusters in the final clustering result. Obviously, this is meaningless, so a point within a certain range around the found cluster center may not be the cluster center of the next cluster. When selecting the best initial center of the next cluster, these points need to be excluded first. The specific implementation process is as follows.

If we know the final result (m_1, m_2, ..., m_k) of the clustering problem of (k-1) clusters, we get all the clusters (c_1, c_2, ..., c_{k-1}) with (m_1, m_2, ..., m_{k-1}) as the cluster center. First we calculate the average distance r_t between all data points in each cluster and the cluster center, where t = 1, 2, ..., k - 1.

\[ r_t = \frac{\sum_{x \in C_t} ||m_t - x||^2}{n_t} \]  
(6)

where n_t is the number of all data points in C_t.

We refer to the parameter \( \alpha_t = (\max_{x \in C_t} (||m_t - x||^2))/r_t \) of [44], obviously \( \alpha_t > 1 \). Let \( \beta_t = \varepsilon(\alpha_t - 1) \), where \( \varepsilon \) is a sufficiently small positive number. Then we introduce a parameter \( \lambda_t = 1 + \beta_t \cdot (k - 1) \); obviously \( \lambda_t \geq 1 \), (t = 1, 2, ..., k - 1).

All points in the area with \( r = r_t \cdot \lambda_t, (t = 1, 2, ..., k - 1) \) as the radius centered on m_t are removed, and the initial cluster center of the next cluster is selected from the remaining points, which will effectively reduce the number of data points that need to be calculated. This reduces the amount of calculation. In the incremental method, as the k increases, the structure of the obtained cluster is also more and more stable, so the value of the parameter \( \lambda_t \) can be increased accordingly to exclude more points.

4.3.2. Spectral Clustering Algorithm. In the spectral clustering algorithm, we first need to solve the similarity matrix, in which we define the overlap degree matrix P as the similarity matrix. For arbitrary P_{ij} = P_{ji} and P_{ii} = 0; that is, diagonal element is 0. Then the overlapping degree matrix can be expressed as

\[ P = \begin{bmatrix} 0 & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & 0 & \cdots & p_{2,n} \\ \vdots & \cdots & 0 & \vdots \\ p_{n,1} & \cdots & p_{n,n-1} & 0 \end{bmatrix} \]  
(7)

where \( p_{i,j} = \exp(-||s_i - s_j||^2/2\sigma^2) \). Then we can calculate the normalized matrix \( D(i,i) = \sum_{j=1}^{n} p_{i,j} \), and Laplacian matrix \( L = D - P \).

We can assume that all the requests are divided into k_1, k_2, two categories, for the classification criteria; this paper uses the standard deviation (NORMALIZED CUT) method. Let q be a vector. The elements q_i are defined as follows:

\[ q_i = \begin{cases} \frac{d_2}{d_1 + d_2}, & i \in k_1 \\ -\frac{d_1}{d_1 + d_2}, & i \in k_2 \end{cases} \]  
(8)

Among them, \( D_i \) is the sum of all overlaps in k_1, plus Cut (k_1, k_2), \( D_2 \) is the sum of all overlaps in k_2, plus Cut (k_1, k_2), and \( d = d_1 + d_2 - \text{Cut}(G_1,G_2) \).

Then \( N_{\text{cut}}(k_1,k_2) = q^T L q \); the restrictions are

\[ q^T D q = \sum_{i \in k_1} q_i^2 D_{ii} + \sum_{j \in k_2} q_j^2 D_{jj} = \left(\frac{d_2}{d_1 + d_2}\right)^2 d_1 + \left(\frac{d_1}{d_1 + d_2}\right)^2 d_2 = 1 \]

\[ L(q, \lambda) = N_{\text{cut}}(G_1,G_2) - \lambda (q^T D q - 1) = q^T L q - \lambda (q^T D q - 1) \]

\[ \frac{\partial L(q, \lambda)}{\partial q} = 0 \implies L q = \lambda D q \]
Let $L' = D^{-1/2}LD^{-1/2}$, $q' = D^{-1/2}q$

$: D$ is diagonal matrix

$D^{1/2}D^{1/2} = 1 \implies L'q' = \lambda q$

(9)

At this time, the Laplacian matrix for the normalized (Normalized Laplacian, the diagonal elements are all 1) $L' = D^{-1/2}L D^{-1/2}$ eigenvalues and the corresponding eigenvectors. Because the eigenvalues of $L$ and $L'$ are the same, the relation between the eigenvectors is $q' = D^{1/2}q$, so we can get the eigenvectors corresponding to the eigenvalues of $L'$, and finally we can get the $q$ by multiplying $D^{-1/2}$.

4.3.3. Improved Spectral Clustering Algorithm (ISCM). Based on the improved k-means algorithm, we propose an improved spectral clustering algorithm (ISCM). The algorithm not only solves the problem of initial value, but also realizes the secondary clustering. As described above, according to the QoS condition, we consider the parameters $\delta$. When the algorithm is executed, we can through judge the size of the parameters whether it needs secondary clustering. If the current QoS satisfies the demand of users for resource scheduling after the algorithm is executed, there is no need to recluster. Next, the implementation steps of spectral clustering optimization scheduling algorithm are introduced.

The detailed steps of the scheduling algorithm are as follows:

1. Transforming a set $S = \{ s_1, s_2, \ldots, s_n \}$ into a set $\hat{S} = \{ s_1, s_2, \ldots, s_n, s_{n+1}, \ldots, s_{n+A} \}$.
2. According to the overlap degree between the elements in the set $\hat{S}$, the overlap degree matrix $P$ is calculated, and the value of $c_{ij}$ is calculated according to the formula $c_{ij} = 1 - \beta(1 - p_{ij})$.
3. Constructing Laplacian matrix $L$ and computing eigenvalues of matrix $L$.
4. Constructing matrix $R$ by eigenvectors corresponding to eigenvalues.
5. Computing $k_{\text{max}}$ and $k_{\text{min}}$.
6. For $k \in [k_{\text{min}}, k_{\text{max}}]$. Using improved k-means method to cluster row vectors of matrix $R$.
7. Computing delay $E$, QoS conditions $\delta'$, and system resource utilization $U$, according to formula (3): $U = \sum_{r=1}^{k} t_r / KT$.
8. If $\delta' \leq \delta$, the loop is stopped; otherwise, go back to the previous step and restart.
9. The scheduling strategy can be obtained according to $k$ cluster.

The flowchart of the ISCM algorithm is shown in Figure 4.

4.4. Resource Scheduling

4.4.1. Scheduling Process. In the resource scheduling process of edge computing, as shown in Figure 5, the end user submits the service request to the edge micro data center (EMDC) through the edge computing. The resource monitor is set up in EMDC, and the dynamic resource monitor will detect the status information of the available resources at any moment and save the status information of these resources in EMDC. The ISCM algorithm clusters the service request sets, sorts them into several different sets of requests, and then schedules the resources. Finally, the resource allocation of these requests is arranged.

4.4.2. Pseudocode of ISCM Algorithm. Through the above steps, the resource scheduling problem is transformed into a
clustering problem, and the resource scheduling scheme can be obtained according to the clustering results. ISCM algorithm can effectively solve the resource scheduling problem in edge computing.

The pseudocode of ISCM algorithm is as in Algorithm 1.

4.5. Algorithm Correctness and Performance Analysis. The algorithm consists of three stages; the first stage calculates the overlap degree between the user service request sets; each service request contains the start time and end time; the overlapping degree matrix can be obtained according to the overlap in time interval. In the second stage, the overlap degree matrix is diagonalized, and the Laplacian matrix is established by using the normal cutting method, and the eigenvector is obtained. In the third stage, we use the myk-means algorithm to cluster, and on the basis of the original k-means algorithm, the initial clustering center selection algorithm is added, repeatedly traversing the search data samples to find k best initial clustering centers. Then, according to the initial cluster centers, the k cluster is obtained to form scheduling strategy. Throughout the above three stages, the degree of time overlap between users is regarded as the similarity between data samples, and the scheduling problem is transformed into clustering problem. We classify the large overlap degree requests into one class and divide them into different classes with small overlap degree. The clustering algorithm can make the similarity among samples the largest, and the similarity between samples is minimum, so our algorithm is correct.

Theorem 4-1 ISCM algorithm can complete the resource scheduling in edge computing; the time complexity is $O(\text{nick})$. $n$ represents the number of data samples, $c$ represents the number of times that the initial cluster center searches, and $k$ represents the number of clusters.

In large-scale datasets, the number of searches $c$ is much less than the number of samples, so the time complexity of the proposed algorithm avoids sinking into worst-case $O(n^2)$.

According to the common performance metrics of clustering algorithm, as shown in Table 1, this paper compares with ISCM algorithm, traditional spectral clustering algorithm, and k-means algorithm.

5. Simulation Results

Our simulation experiments are based on the MATLAB platform, assuming the known service quality (QoS) conditions $\delta = 0.1$, $T = 60$, $m = 5$ (which means that each user can apply for 5 computing units at most). The busy time of the 5 computing units in $T$ is denoted as $T_1$, $T_2$, $T_3$, $T_4$, and $T_5$, and their values, respectively, are 35, 20, 30, 39, and 53.

According to the starting time and completion time of each request, denoted as $T_1$ and $T_2$, and the number of units that need to be occupied, represented by the letter $a$, the request is represented as data point in three-dimensional...
Table 1: Performance comparison of algorithms.

<table>
<thead>
<tr>
<th>Performance metrics</th>
<th>ISCM algorithm</th>
<th>Traditional spectral clustering algorithm</th>
<th>k-means algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extendibility</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Data types that can be processed</td>
<td>Numerical and non-numerical type</td>
<td>Numerical and non-numerical type</td>
<td>Numerical type</td>
</tr>
<tr>
<td>The clustering shape can be found</td>
<td>Arbitrary</td>
<td>Arbitrary</td>
<td>Convex and spherical type</td>
</tr>
<tr>
<td>Domain knowledge dependence on input parameter</td>
<td>Small</td>
<td>Large</td>
<td>Large</td>
</tr>
<tr>
<td>Sensitivity to noise</td>
<td>Sensitive</td>
<td>Sensitive</td>
<td>Sensitive</td>
</tr>
<tr>
<td>Sensitivity to input order</td>
<td>General</td>
<td>General</td>
<td>Sensitive</td>
</tr>
<tr>
<td>The ability to process high dimensional data</td>
<td>High</td>
<td>High</td>
<td>Lower</td>
</tr>
<tr>
<td>Time complexity</td>
<td>O(nck)</td>
<td>O(knt)</td>
<td>O(knt)</td>
</tr>
</tbody>
</table>

According to the diversity of big data in actual situation, we choose two kinds of data sample set. For the first sample set, we use k-means algorithm and myk-means algorithm, respectively. For the second sample set, we use the traditional spectral clustering algorithm and ISCM algorithm, respectively; at the same time, it is compared with k-means algorithm.

When using the traditional k-means clustering algorithm, the experimental results are shown in Figures 7 and 8; the X axis represents the start time of the service request, the Y axis represents the finish time of the service request, and the Z axis represents the number of computing units required for the service request. Here we set up 5 computing units at most that users can apply. It is not difficult to find from Figure 8 that the same color is the same class, the traditional k-means algorithm clustering results are wrong, and this is because the clustering results depend on the initial clustering center.

Figure 9 shows the results of the algorithm using myk-means. As shown in Figure 9, the same color is as the same class, the time overlap between the same classes is large, and users have to wait for resources until the resource is idle, so it reduces customer satisfaction easily. Different types of requests can respond simultaneously, because there is no time overlap, and edge computing allocates resources to them simultaneously, which can avoid resource contention, thus it forms scheduling policy. Different categories of requests have different colors; moreover, different categories of requests are combined into a scheduling set in the scheduling process, which can reduce the overlap degree and service delay and improve QoS and resource utilization.

From the comparison of two algorithms under the experimental results, we can conclude that myk-means clustering algorithm used in this paper is superior to the traditional k-means algorithm, myk-means clustering algorithm selects the appropriate initial cluster center, and it can converge to a better local optimal solution and obtain more accurate resource scheduling.

In Figure 10, the blue line represents the running time of traditional spectral clustering algorithm, and the red line represents the running time of ISCM algorithm. As can be seen from Figure 10, by improving the spectral clustering algorithm and reducing the amount of data calculation, the...
execution efficiency of the ISCM algorithm can be effectively improved. In addition, as the scale of requests continues to increase, the time-growth trend of the ISCM algorithm is slower than the traditional spectral clustering algorithm.

In order to show that the ISCM algorithm is better than k-means algorithm and traditional spectral clustering algorithm in clustering accuracy and applicability scope, the experimental results obtained in this paper are shown in Figures 11 and 12.

In Figure 11, we can see that the clustering results of traditional spectral clustering algorithm are not stable, due to the clustering results depending on the selection of the initial clustering center; when poor initial clustering center is selected, the clustering of traditional spectral clustering algorithm produce the result is not stable.

In Figure 12, the traditional k-means algorithm can not solve the clustering problem, and our algorithm can get stable solution to the problem of clustering. In addition, when the QoS conditions do not meet the needs of users, the proposed algorithm will be reclustering until meeting the user needs. In summary, the algorithm of this paper is superior to the traditional k-means algorithm and traditional spectral clustering algorithm in solving the problem of edge computing resource scheduling. Comparing with Figure 11, it is not difficult to find that the clustering results obtained by ISCM algorithm are ideal.

6. Conclusions

In this paper, we analyze the user-oriented resource scheduling problem in the edge computing of the Internet of Things and propose the ISCM algorithm to solve the scheduling problem in edge computing. In edge computing, users can simultaneously apply a variety of resources, which results in the overlap in time interval and increases the difficulty of resource scheduling problem. Firstly, we transform the problem of overlap degree in resource scheduling into clustering problem. Secondly, we carry on the overlap degree matrix transformation and the eigenvector clustering. Finally, carry out resource scheduling for the categories with different colors. Experiments show that the scheduling algorithm can stabilize the clustering results under the guaranteed QoS condition, and it can obtain the higher resource utilization rate. In this problem, we assume that each computing unit
Figure 11: Clustering results comparison of traditional spectral clustering algorithm.

Figure 12: Clustering results comparison of ISCM algorithm.
has the same calculation ability, and the future research work should focus on the difference between the calculation ability of different computing units.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (61672321 and 61771289), the Shandong Provincial Graduate Education Innovation Program (SDYY14052 and SDYY15049), Qufu Normal University Science and Technology Project (xkj201525), Shandong Province Science and Technology Planning Project (J16LN15), the Shandong Provincial Specialized Degree Postgraduate Teaching Case Library Construction Program, and the Shandong Provincial Postgraduate Education Quality Curriculum Construction Program.

References


Fault tolerance is of great importance for big data systems. Although several software-based application-level techniques exist for fault security in big data systems, there is a potential research space at the hardware level. Big data needs to be processed inexpensively and efficiently, for which traditional hardware architectures are, although adequate, not optimum for this purpose. In this paper, we propose a hardware-level fault tolerance scheme for big data and cloud computing that can be used with the existing software-level fault tolerance for improving the overall performance of the systems. The proposed scheme uses the concurrent error detection (CED) method to detect hardware-level faults, with the help of Scalable Error Detecting Codes (SEDC) and its checker. SEDC is an all unidirectional error detection (AUED) technique capable of detecting multiple unidirectional errors. The SEDC scheme exploits data segmentation and parallel encoding features for assigning code words. Consequently, the SEDC scheme can be scaled to any binary data length “n” with constant latency and less complexity, compared to other AUED schemes, hence making it a perfect candidate for use in big data processing hardware. We also present a novel area, delay, and power efficient, scalable fault secure checker design based on SEDC. In order to show the effectiveness of our scheme, we (1) compared the cost of hardware-based fault tolerance with an existing software-based fault tolerance technique used in HDFS and (2) compared the performance of the proposed checker in terms of area, speed, and power dissipation with the famous Berger code and m-out-of-2m code checkers. The experimental results show that (1) the proposed SEDC-based hardware-level fault tolerance scheme significantly reduces the average cost associated with software-based fault tolerance in a big data application, and (2) the proposed fault secure checker outperforms the state-of-the-art checkers in terms of area, delay, and power dissipation.

1. Introduction

Big data is promising for business applications and is rapidly increasing as an important segment of the IT industry. Big data has also opened doors of significant interest in various fields, including remote healthcare, telebanking, social networking services (SNS), and satellite imaging [1]. Failures in many of these systems may represent significant economic or market share loss and negatively affect an organization’s reputation [2]. Hence, it is always intended that whenever a fault occurs, the damage done should be within an acceptable threshold rather than beginning the whole task from scratch, due to which fault tolerance becomes an integral part in cloud computing and big data [3]. Fault tolerance prevents a computer or network device from failing in the event of an unexpected error [2]. A recent study [4] showed that the cost of fault tolerance in cloud applications with high probability of failure and network latency is around 5% for the range of application sizes, hence providing improved performance at a lower cost.

The fault tolerance schemes in popular big data frameworks like Hadoop and MongoDB are composed of some sort of data replication or redundancy [5, 6]. MongoDB replicates its primary data in secondary devices. In a faulty event, the data is recalled from the secondary or the secondary temporarily acts as a primary. Fault tolerance in Hadoop relies on multiple copies of data stored on different data nodes. Although replication schemes allow complete data recovery,
they consume a lot of memory and communication resources. Hence, in recent years, many researchers have proposed fault tolerance algorithms for improved data recovery, effective fault detection, and reduced latency in big data and cloud computing [2, 5–10]. All of which detect fault at the software (SW) level. Even though faults propagated due to transient errors in hardware are also detected by these schemes, and software-based techniques are more flexible, the amount of data required to process to detect a fault costs a lot more than hardware- (HW-) based fault tolerance schemes. A recent study [11] investigated the cause of data corruption in a Hadoop Distributed File System (HDFS) and found that when processing uploaded files, HW errors such as disk failure and bit-flips in processor and memory generate exceptions that are difficult to handle properly. Liu et al. [7] implemented some level of HW-based fault tolerance by modelling CPU temperature to anticipate a deteriorating physical machine. Liu et al. [7] proposed the CPU temperature monitoring as an essential step for preventing machine failure due to overheating as well as for improving the data center's energy efficiency.

Parker [12] discussed how in many cases the faults are a direct consequence of tightly integrating digital and physical components into a single unit at a sensor or field node. In fact, many modern systems rely so heavily on digital technology that the reliability of the system cannot be decomposed and partitioned into physical and SW components due to interactions between them. There is a cost associated with the storage, transmission, and analysis of these higher-dimensional data. Furthermore, many of the SW-based approaches are simulation intensive, which may lead to broad implementation challenges. To overcome some of these challenges, he suggested that onboard, embedded processing will be a practical requirement.

Transient errors in HW, if propagated, may cause chain reaction of errors at the SW layer, causing potential failure at the node/server level. Detection at the HW level requires less computation time (as low as single clock cycle) as compared with detection at the SW level (several machine cycles), while a simple recovery mechanism called recomputation at the HW level can save a lot of data swapping and signaling at the SW level. As discussed in [13], big data has created opportunities for semiconductor companies to develop more sophisticated systems to cover the challenges faced in big data and cloud computing, and a trend towards integration of more functions onto a single piece of silicon is likely to continue. Also, due to advances in semiconductor processing, there has been a reduction in the cost of digital components [12]. For these reasons, we propose the detection of transient faults, as they occur in HW, through a HW-based fault tolerance scheme, while the SW-based fault tolerance stays at the top level as a second check for HW errors and first check for SW errors. As a result, the transient errors that arise in HW are mostly taken care of by lightweight processing at the HW level with little overhead (in terms of area, power, and delay), saving tremendous computation resources at the system level. The potential for catastrophic consequences in big data systems justify the overhead incurred due to HW-based fault tolerance method.

On the other hand, fault tolerance has also become an integral part of very large-scale integration (VLSI) circuits, where downsized, large-scaled, and low-power VLSI systems are prone to transient faults [14]. Transient faults or soft errors are transient-induced events on memory and logic circuits caused by the striking of rays emitted from an IC package and high energy alpha particles from cosmic rays [14–18]. Also, in multilevel cell memories like NAND Flash memories, these errors are mostly caused by cell-to-cell interference and data retention errors [19]. Physical protection such as shielding, temperature control, and grounding circuits are not always feasible; hence, in such cases, concurrent error detecting (CED) methods are employed for protection against these errors. Since CED circuits add to the overall area and delay of the system, the selection of appropriate error detecting, and even error correcting, circuits for a particular application leads to an efficient design [18]. It has been reported that the biggest portion of errors that occur in VLSI circuits and memories are related to unidirectional errors (UE) [19–21] because these errors shift threshold voltage levels to either the positive or negative side [22], causing the circuit node logic from “0” to “1” or from “1” to “0,” but not both at the same time.

Many all unidirectional error detection (AUED) schemes have been proposed and implemented, among which the Berger code technique [23] is agreed to be the least redundant. With the ability to detect single- as well as multiple-bit unidirectional errors, this technique provides error detection by simply summing the logic 0’s (a B0 scheme) or 1’s (a B1 scheme) in the information word, expressing its sum in binary. If the information word contains “n-bits,” then a Berger code will require \( \lceil \log_2(n + 1) \rceil \)-bits. A Berger code checker employs a 0’s (or 1’s) counter circuitry for reencoding the information word to check bits and then compares it with the preencoded check bits using a two-rail checker [23]. A chain of adders and a tree of two-rail checkers are required to design these checker circuits [23], where area and latency increase drastically as data length increases.

An m-out-of-n code is one in which all valid code words have exactly “m” 1’s and “n-m” 0’s. These codes can also detect all unidirectional errors when \( n = 2m \). This condition not only increases the code size, but also the checker’s area. Cellular realization of an m-out-of-2m code circuit was deemed by Lala [24] as more area- and delay-efficient than the previous implementations.

Given the importance of fault tolerance at the HW level in big data and cloud computing applications, in this paper, we present a fault secure (FS) SEDC checker used with SEDC codes [25]. An FS checker has the ability to safely hide or self-check (detect) its own faults as they occur in its circuitry. The SEDC partitions the input data into smaller segments (2, 3, and 4 bits) and encodes them in parallel. This unique scaling feature makes the system faster and less complex to design for any binary data length. The FS SEDC checker inherits all these features of SEDC codes (i.e., simple scalability, constant latency, and less power dissipation) which suits its implementation in online fault detection in processors, cache memories, and NAND Flash-based memories for big data.
applications. The major contributions of this paper are as follows:

1. We propose HW-level fault tolerance for circuits designed to process big data and cloud computing applications.

2. In order to show the effectiveness of the proposed HW-level fault tolerance scheme in a big data scenario, we compare the cost associated with and without the proposed fault tolerance scheme and present results that show a significant reduction in the overall cost of fault tolerance in big data when the proposed HW-based fault tolerance scheme is applied.

3. We also present a novel FS SEDC checker for use with SEDC-based HW-level fault tolerance systems.

4. In order to prove the superiority of the FS SEDC checker presented in contrast with state-of-the-art AUED checkers, we show that the FS SEDC checker achieves state-of-the-art performance in terms of area, delay, and power dissipation.

The rest of the paper is organized as follows. We present an overall system diagram of the proposed HW-level fault tolerance system in Section 2. We give a brief mathematical foundation of the SEDC scheme and an example to encode logical circuits using SEDC in Section 3. Design details of the FS SEDC checker are described in Section 4. The proposed checker is shown to be FS through the fault testing methods; and its area, delay, and power comparison with state-of-the-art are derived in Section 5. We compute the fault coverage of the proposed SEDC-based fault tolerance system and present the experimental details and results in Section 5. To show the effectiveness of the proposed method in big data and cloud computation applications, we also perform a cost-performance analysis of fault tolerance at the SW level versus HW level in Section 5. Finally, we conclude the paper in Section 6.

2. Introduction to the Overall System

Figure 1 shows the main components of an error detecting codes based HW-level fault tolerance. The functional circuit consists of two subcircuits: an information symbol generator (ISG) and a check symbol generator (CSG). These two circuits do not share any logic. The ISG takes input D and performs some operation G and produces output G(D). The CSG is a carefully chosen logic function that acts as the encoder and generates check bits S using the same input D, such that S = \( \phi(G(D)) \), where \( \phi \) denotes the particular coding function. The checker normally contains another encoder that reencodes the information bits G(D) into \( S' = \phi(G(D)) \) and then compares both S and S'. A mismatch between S and S' is treated as an error, which is indicated by the error indication or verification signal V.

The checker shown in Figure 1 plays a vital role in the overall fault tolerance system. The checker must exhibit a self-checking property or failsafe property to make sure that the whole system is fault secure (FS). If the checker is both self-checking and failsafe, the overall system is said to be as totally self-checking (TSC). In order to formally define these properties, let us consider the output of the functional circuit shown in Figure 1 to be represented by \( G(D) = G(x, f) \), where \( x \) is the input and \( f \) is the fault, and then in fault-free operation, i.e., \( f = \emptyset \), the output can be represented by \( G(x, \emptyset) \). Also, consider the input code space \( D \subseteq X \), output code space \( S \subseteq Y \), and an assumed fault set \( F \); then according to the definition of totally self-checking (TSC), \( G \) is

(1) self-testing if for each fault \( f \) in \( F \) there exists at least one input code \( d \in D \) that produces a noncode output; i.e., \( \forall f \in F \exists d \in D \ni G(e, f) \notin S \),

(2) fault secure (FS) if for all faults \( f \) in \( F \), and all code inputs \( d \in D \), the output is either correct or is a noncode word; i.e., \( \forall f \in F \text{ and } \forall d \in D, G(e, f) = G(e, \emptyset) \text{ or } G(e, f) \notin S \).

In the proposed SEDC-based HW-level fault tolerance system, the CSG circuit is realized by an SEDC check symbol generator (SCSG) circuit, which generates the SEDC code words corresponding to the information bits G(D). We presented a realization of an SEDC encoded SCSG circuit in [27], i.e., an SEDC encoded arithmetic logic unit (ALU) of a microprocessor. The SEDC encoded ALU circuit (SCSG) computes the SEDC codes corresponding to the output of the
3. Scalable Error Detection Coding (SEDC) Scheme

The Scalable Error Detection Coding scheme [25] is an AUED scheme formulated and designed in such a way that only the resultant circuit area is scaled, while its latency depends on a small portion of the input data (explained later).

For any binary data \( D \) of length \( n \)-bits represented as \( (D_{n-1}, \ldots, D_2, D_1, D_0) \) with \( D_i \in \{0, 1\} \) for \( 0 \leq i \leq n - 1 \), two parameters, \( a \) and \( b \), are computed using

\[
a = \frac{n - \max(b)}{3} \quad (1)
\]

where parameter \( a \) can only take a positive integer value, i.e., \( a \in \mathbb{Z}^+ \), and parameter \( b \) is in \( \{2, 3, 4\} \). Satisfying the condition for parameter \( a \), the maximum possible value for parameter \( b \) is selected. The SEDC code word \( S \) is represented as \( (S_{m-1}, \ldots, S_j, S_{j+1}, S_{j+2}, S_0) \) with \( S_j \in \{0, 1\} \) for \( 0 \leq j \leq m - 1 \), where \( m \) denotes the length of the SEDC code word and is computed by

\[
m = \left\lceil \log_2 (n + 1 - 3a) \right\rceil + 2a \quad (2)
\]

After computing the values for parameters \( a \) and \( b \), the SEDC code \( S \) for binary data \( D \) is computed. SEDC is designed to generate codes basically for 2-, 3-, and 4-bit data and is accordingly referred to as the SEDC_2, SEDC_3, and SEDC_4 scheme, respectively. It is then extended for any integer values of \( n \), as shown in Figure 2(a).

3.1. SEDC_2 Code. A two-dimensional (2D) illustration of a 2-bit SEDC (SEDC_2) scheme is shown in Figure 2(b), where nodes represent data words, and their corresponding code words are written in brackets.

The SEDC coding scheme assigns code words to different data words with a unique criterion. Whenever there is a change of a bit (or bits) in a data word from “1” \( \rightarrow \) “0,” as shown with a bold arrow in Figure 2(b), the change is reflected in the code word in the opposite way; i.e., the code changes from “0” \( \rightarrow \) “1,” as shown with the dashed arrow in Figure 2(b), and vice versa. Equation (3) is used to assign 2-bit code words \( S_1, S_0 \) to the 2-bit data words \( D_1, D_0 \). Clearly, we can interchange the bit positions of \( S_1 \) and \( S_0 \) for another variant of SEDC_2 codes. This will not affect the code characteristics.

\[
[S_1 : S_0] = S_{\text{EDC}_2}(D_1, D_0) = [\text{XNOR}(D_1, D_0) : \text{NAND}(D_1, D_0)] \quad (3)
\]

In (3), \( [S_1 : S_0] \) represent the concatenated SEDC code bits, XNOR and NAND are the logical operations, and SEDC_2 is the basic coding scheme.

3.2. SEDC_3 Code. SEDC_3 code for 3-bit data is computed using (4) as follows:

\[
[S_1 : S_0] = S_{\text{EDC}_3}(D_2, D_1, D_0) = \begin{cases} 
S_{\text{EDC}_2}(D_1, D_0), & \text{if } D_2 = 0 \\
S_{\text{EDC}_2}(D_1, D_0), & \text{if } D_2 = 1 
\end{cases} \quad (4)
\]

where the bar sign (e.g., \( \overline{D_1} \)) in (4) represents the logical NOT operation.

Figure 3 shows a 3D cube, illustrating the unidirectional error detection mechanism of SEDC_3 codes. The same notations are used in Figure 3 as in Figure 2(b). The dashed side of the cube represents the embedded SEDC_2 coding scheme in SEDC_3. Note that when there is a 2-bit unidirectional change in data word “00” \( \rightarrow \) “11” (the two MSBs changing from “00” \( \rightarrow \) “11”), the code changes in the opposite direction (the least significant bit of the code changes from “1” \( \rightarrow \) “0”). In a similar way, the SEDC_m scheme detects \( n \)-bit or all unidirectional errors in the data word \( D \).

Figure 2: (a) SEDC scheme for given data word, and (b) 2D illustration of SEDC_2 scheme.
3.3. SEDC\(_4\) Code. A SEDC\(_4\) code for 4-bit data is formulated by (5) as follows:

\[
[S_2 : (S_1, S_0)] = SEDC_4 (D_3, D_2, D_1, D_0) = \left[ \overline{D_3} : SEDC_3 (D_2, D_1, D_0) \right]
\]

The MSB of the code word is completely dependent upon the MSB of the data word for SEDC\(_3\): hence, any change in the MSB of the data word is detected. The rest of the three data bits are encoded using the same SEDC\(_3\) scheme.

It can be observed from (3), (4), and (5) that the SEDC\(_3\) is embedded in 3-bit SEDC (SEDC\(_4\)) and consequently in 4-bit SEDC (SEDC\(_4\)) to detect all unidirectional errors in 3-bit and 4-bit data, as shown later. This ability to scale codes is not present in any other concurrent error detecting (CED) coding scheme.

In general, for SEDC\(_n\), the \(n\)-bit binary data is grouped into one \(b\)-bit segment and the \(a\) number of 3-bit segments, and then these segments are encoded using one SEDC\(_a\) and a number/numbers of SEDC\(_3\) modules in parallel, as shown in Figure 2(a). It is noteworthy that each group of data segments and corresponding code segments is independent of each other. This independence makes our scheme scalable and able to detect some portion of bidirectional errors (BE) (discussed in Section 5.3).

If we interchange \(S_1\) and \(S_0\) for SEDC\(_4\) in Figure 3, the corresponding SEDC\(_3\) code is equal to Berger codes for a 3-bit segment, but our way of deriving the SEDC\(_4\) code is a lot different from that of Berger codes. SEDC\(_3\) codes are basically scaled from SEDC\(_2\) codes, and SEDC\(_2\) codes have no commonality with 2-bit Berger codes.

3.4. SEDC-Based HW-Level Fault Tolerance System Example. In order to illustrate the designing of a HW-level fault tolerance system using the SEDC scheme, we take the example of a 4-bit adder. Let us consider that this 4-bit adder is a part of a processor which processes big data applications, and we want to make this 4-bit adder fault tolerant against transient errors that arise in its circuitry, so the general HW-level fault tolerance system diagram shown in Figure 1 will be converted to the one shown in Figure 4. As shown in Figure 4, the 4-bit adder acts as an ISG and its equivalent SEDC encoder acts as a CSG. The SEDC encoder or CSG can be implemented using (6) as follows:

\[
[S_3 : S_0] = SEDC (A_{[3:0]}, B_{[3:0]} + C_m)
\]

As the output of 4-bit adder is a 5-bit value, hence the equivalent SEDC code has a 4-bit value according to (2). We used Altera’s Quatrus II software to synthesize the 4-bit adder (ISG), SEDC encoder (CSG), and the SEDC checker shown in Figure 4 and utilized the synthesized circuit for computing the fault coverage of the SEDC scheme, which is presented in Section 5.3. In the next section, we present the proposed FS SEDC checker, which completes the overall proposed SEDC-based HW-level fault tolerance system.

4. The FS SEDC Checker

As shown in Figure 4, the FS SEDC checker takes \(n\)-information bits and \(m\)-SEDC check bits from the functional unit. The FS SEDC checker is also composed of one \(b\)-bit FS SEDC checker and \(a\) sets of 3-bit FS SEDC checkers. With 1-, 2-, and 3-bit FS SEDC checkers, the output can be directly used as an error indication signal, but for \(n > 3\), one level of wired-AND-OR logic gates is used to combine all the output of subblocks of FS SEDC checkers and generate the 2-bit error indication signal. Subsections discuss logic and circuit diagrams for primitive FS SEDC checkers (SEDC\(_1\), SEDC\(_2\), SEDC\(_3\), and SEDC\(_4\) checkers) which can be used to scale the SEDC checker to an \(n\)-bit FS SEDC checker (i.e., an FS SEDC\(_n\) checker).

4.1. The FS SEDC\(_1\) Checker

Table 1 shows the logic for a 1-bit SEDC (FS SEDC\(_1\)) checker. The valid input code words are “10” and “01” and the valid output code word is “10.” \(G_0\) denotes the 1-bit information word that is the output of ISG, and \(S_0\) denotes the 1-bit SEDC check bit generated by the SEDC check symbol generator (SCSG). \(V_1\) and \(V_0\) signals are generated by the circuits shown in Figure 5(a).

4.2. The FS SEDC\(_2\) Checker

\[
[V_1 : V_0] = \left[ S_1, (G_1 + G_0) (S_0 + G_1 G_0) : \right. \\
\left. (G_1 + G_0 + S_0) (S_1 + G_1 G_0 S_0) \right]
\]

In Figure 5, the symbols P1-P13 and N1-N13 represent the PMOS and NMOS transistors, respectively, and Vss represents the voltage supply. For simplicity, we used the CMOS-based implementation of SEDC checker circuits. Any other technology can be used to design these circuits, but the underlying algorithm, i.e., SEDC, will remain the same.

4.3. The FS SEDC\(_3\) Checker

Figure 6(a) shows the block diagram and the logic for a 3-bit FS SEDC checker. Three-bit data \(G_2 G_1 G_0\) from the ISG and 2-bit SEDC check bits \(S_3 S_2 S_1\) from the SCSG are first converted to \(G_1 G_0\) and \(S_1 S_0\), respectively, and then are checked using the same 2-bit FS SEDC checker, as shown in Figure 6(a). When the \(G_2\) bit is “1,” \(G_1 G_0\) and \(S_3 S_2 S_1\) are inverted, whereas if \(G_2\) is “0,” then \(G_1 G_0\) and \(S_3 S_2 S_1\) remain the same. As the outputs of the XOR gates are fed to the FS SEDC\(_3\) checker, any error in the XOR gates is detected. This makes the overall 3-bit SEDC checker FS.

<table>
<thead>
<tr>
<th>(G_0)</th>
<th>(S_0)</th>
<th>(V_1)</th>
<th>(V_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
4.4. The FS SEDC₀ Checker. A 4-bit FS SEDC checker consists of one FS SEDC₁ checker and one FS SEDC₃ checker, as shown in Figure 6(b). Both SEDC₁ and SEDC₃ checkers generate 2-bit output V₁V₀. Because the valid code word is “10,” to make sure that both checker units generate the “10” output during error-free operation, we “AND” the V₁ output-bit of the FS SEDC₁ checker with the V₀ output-bit of the FS SEDC₃ checker. Also, we “OR” the V₀ output-bits of both FS SEDC checkers using wired logic gates. We checked and confirmed by fault simulation that wired-AND and wired-OR gates are also FS for single faults (stuck-at-0, stuck-at-1, transistor-stuck-on, and transistor-stuck-off).

4.5. The FS SEDCₙ Checker. Like the SEDC code generator, the FS SEDC checker also consists of multiple 1-, 2-, and 3-bit FS SEDC checkers, depending upon the value of a and b from (1). For example, if n = 8 bits, then (1) ⇒ a = 2 and b = 2. This requires one FS SEDC₂ checker and two FS SEDC₃ checkers to realize an 8-bit FS SEDC checker.

The area of wired-AND-OR gates will also definitely increase as n is increased. Figure 7 shows the block diagram of an n-bit FS SEDC checker. For n = 8 bits, there will be total of three FS SEDC checkers, each with 2-bit output; hence, a 3-input wired-AND and a 3-input wired-OR gate is required to compare all V₁ and V₀ bits. In general, for n-bit input, there are “a + 1” FS SEDC checkers, each with 2-bit output. So we require “k = 2 × (a + 1)”-input wired-AND and wired-OR gates. With each increasing input to the wired-AND-OR network, one extra transistor is required by each of the wired gates. This causes the circuit to expand width-wise; hence, the latency of the wired logic remains constant for any value of n.

The size of the load transistor driving these wired-AND and -OR gates will also increase with increasing input, so we consider the maximum fan-in of one gate as equal to 4. For k > 4, an extra load transistor is connected in parallel. Generally, for k inputs we require r = [k/4] load transistors. A total of k + r transistors is required to design the k-input wired AND-OR network with a constant latency of 1 transistor.

5. Experiments and Results

In this section, we present the experiments we conducted on the proposed FS SEDC checker and the overall proposed SEDC-based HW-level fault tolerance system. The results of each experiment are given along with the experimental details in the subsections below.

5.1. Fault Test on FS SEDC Checker. The FS SEDC₁, SEDC₂, SEDC₃, and SEDC₄ circuits in our paper were tested for stuck-at-0, stuck-at-1, transistor-stuck-ON, and transistor-stuck-OFF faults. We assume a single-fault model where faults occur one at a time, and there is enough time between detection of the first fault and the occurrence of another fault [29]. In Table 2, we provide a summary of fault analysis of an SEDC₁ checker circuit. We applied one fault at a time in
the circuit of Figure 5(a) and observed the output. In single-fault operation, the circuit either produced the correct output or never produced any invalid code words (exhibiting FS property), as shown in Table 2.

Case 1 (transistor stuck ON). In Table 2, we show all six cases of transistor stuck ON faults (one at a time). For the cases with N3 or N4 stuck ON, the circuit shows fault detection by one input code combination (represented with * symbol), and hence, the circuit is self-testing, whereas other cases showed that the circuit is fault secure as well as code disjoint.

Case 2 (transistor stuck OFF). In Table 2, all six cases for transistor stuck OFF faults are shown. In cases where N1 or N2 was stuck OFF, the circuit demonstrates the self-testing property (represented with * symbol) and for the rest of the cases, the circuit is fault secure.

Case 3 (input stuck at 0). When input G0 or S0 is stuck at 0, the circuit demonstrates the self-testing property; otherwise, it remains fault secure.

Case 4 (input stuck at 1). When input G0 or S0 is stuck at 1, the circuit shows the self-testing property; otherwise, it remains fault secure.

There is one case where the output becomes floating (i.e., P3 or P4 stuck OFF). In either case, if we consider the floating voltage as logic high, then the circuit is fault secure, and if we consider the floating voltage as logic low, then the circuit is self-testing. Hence, we can say that the circuit in Figure 5(a), which is a 1-bit SEDC checker, is FS. Similar analysis was carried out when testing 2-, 3-, and 4-bit SEDC checkers, and we found that all these checkers are FS.

5.2. Area, Delay, and Power Comparison. In this section, we compare the area and delay of TSC Berger, FS SEDC, and m-out-of-2m code checkers. We use two possible TSC Berger checker implementations from Piestrak et al. [23] and Pierce Jr. and Lala [26], with the m-out-of-2m code checker from Lala [24] for comparison. For the sake of fairness, the area overhead was measured in terms of the number of equivalent
transistors. We made use of the assumptions by Smith [30] to translate gate-level circuits to transistor-level circuits.

Before comparison, we illustrate the functional dissimilarities of the three checkers with the help of Figure 8. Figure 8(a) shows the general block diagram of a TSC Berger code checker. For all the information symbols that the ISG of the functional circuit can produce in normal operation, the check symbol complement generator (CSCG) outputs \((S_B)\) correspond to the bit-by-bit complement of the expected check symbol \(S_B\). The TSC two-rail checker validates that each bit of \(S_B\) is the complement of corresponding bit of \(S_B\). As the size of the input data increases, the length of check symbol \(S_B\) also increases, resulting in a longer length for the TSC two-rail checker tree, and hence the resulting delay.

A general block diagram of a TSC m-out-of-2m code checker is shown in Figure 8(b). The checker takes the information bits and check bits \(S_W\) and partitions them into two parts. The numbers of 1’s, i.e., the weight, of both parts are mapped to a pair of values which in binary belongs to a code, in most cases a two-rail code. The checker consists of a cellular structure of AND-OR gates as given by Lala [24].

Figure 8(c) depicts the general block diagram for an FS SEDC checker that resembles the structure of an m-out-of-2m code checker and differs from a Berger code checker. The FS SEDC checker block receives the information and check bits from the functional unit. If the input data length increases, the size of the FS checker block increases width-wise. The FS SEDC block contains "a + 1" pairs of small SEDC checkers (subblocks). Each subblock of the FS SEDC checker produces "10" as the valid code output. The overall SEDC checker has a final 2-bit output \(S_{10}\), unlike two-rail

![Functional circuit output](image)

**Figure 7:** Block diagram of FS SEDC\(_n\) checker.

**Table 2:** Results of single faults on FS SEDC\(_1\) checker.

<table>
<thead>
<tr>
<th>MOS Plor P2 is stuck ON</th>
<th>MOS P3 or P4 is stuck ON</th>
<th>Input (C_0) stuck at zero</th>
<th>MOS Plor P2 is stuck OFF</th>
<th>MOS P3 or P4 is stuck OFF</th>
<th>Input (F_0) stuck at zero</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0 1 1 0 1 0</td>
<td>1 0 1 0 1 1 0 1</td>
<td>*0 0 1 1 1</td>
<td>1 0 1 0 1 0</td>
<td>0 1 0 1 1 0</td>
<td>*0 0 1 1 1</td>
</tr>
<tr>
<td>0 1 1 0 0 1 1 0</td>
<td>1 0 1 1 1 0</td>
<td>-</td>
<td>1 0 1 1 1 0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0 1 1</td>
<td>*0</td>
<td>1 0 1 1</td>
<td>*0</td>
<td>-</td>
</tr>
<tr>
<td>*0 1 0 0</td>
<td>0 1 1 0</td>
<td>-</td>
<td>0 1 0 0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>*0 1 1 0</td>
<td>0 1 1 0</td>
<td>*0</td>
<td>1 0 1 0</td>
<td>*0</td>
<td>-</td>
</tr>
<tr>
<td>*0 1 1 0</td>
<td>0 1 1 0</td>
<td>-</td>
<td>0 1 0 0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 1 0</td>
<td>-</td>
<td>0 1 0 0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*The cases where circuit shows self-testing property.*
codes, only one of the output combinations "10" is considered a valid code word. A nonvalid checker output "00," "01," or "11" at output $S_{10}$ indicates the presence of a fault in the functional circuit or the FS checker itself. The k-input wired AND-OR network takes the "$a+1$" pairs of output from each SEDC checker subblock and then converts them into a final 2-bit error indication signal $V_S$.

5.1 Fault Test on FS SEDC Checker. Area-optimized realization of TSC Berger code checkers in Piestrak et al. [23] showed less area overhead than m-out-of-2m code checkers, which is apparent from Figure 9. But, if we consider the delay-optimized implementation of the TSC Berger code checker from Pierce Jr. and Lala [26], we see that the TSC Berger code checker requires more area than the FS SEDC and m-out-of-2m codes checkers [24], as shown in Table 3. For clarity, we discretely listed the area overhead offered based on code storage area and code checker area in Table 3. Also listed separately are the area overhead required by the TRC tree for the TSC Berger code checker, the wired-AND-OR network for FS SEDC, and the m-out-of-2m code checker.

For a fair comparison, the extra cost of the code storage area is also taken into account. We assumed that 1-bit storage is implemented by 12-MOS transistors [30]. Table 3 lists the area (in terms of the number of transistors) occupied by FS SEDC, delay-optimized Berger code, and m-out-of-2m code checkers for up to 32-bit data.

The FS SEDC$_n$ checker block shown in Figure 8(c) requires fewer gates, implemented with $[26 + (a \times 50)]$ MOS transistors if “$b = 2$,” $[50 + (a \times 50)]$ MOS transistors if “$b = 3$,” and $[58 + (a \times 50)]$ MOS transistors if “$b = 4$.” The m-out-of-2m code checker implementation of Lala [24] requires $2m^2 - 2m + 2$ gates. The gate-level circuit is also translated to transistor-level circuits using data from Smith [30].

The results show that when scaling a 7-bit 0's counter to an 8-bit 0's counter, 154 extra MOS transistors are required. The m-out-of-2m code checker requires 60 MOS transistors when scaling a 7-out-of-14 checker to an 8-out-of-16 checker, whereas the SEDC checker requires only 18 extra MOS transistors. That is because a 7-bit SEDC checker is implemented with one SEDC$_2$ and one SEDC$_3$ circuit that contain 50 and 58 MOS transistors, respectively (a total of 108 transistors). An 8-bit SEDC checker is implemented using one SEDC$_2$ and two SEDC$_3$ checkers, requiring 26 and 100 (50x2) MOS transistors (a total of 126 transistors). This means that SEDC saves 88% of the number of transistors compared to a Berger code checker [26], and it saves 70% of the transistors when
5.2.2. Delay. As far as delay is concerned, the FS SEDC checker also performs better than Berger and cellular implementations for an m-out-of-2m code checker, as shown in Table 4. For the sake of uniformity, we designed all the basic gates using the same technology transistors (PMOS = $8\mu$/$2\mu$, NMOS = $4\mu/2\mu$) and evaluated the worst-case propagation delay of each circuit.

The SEDC checker shows a constant delay for $n > 3$ bits due to its parallel implementation, whereas the delay in the Berger code checker increases owing to an increase in gate levels (from 6 to 16) in the critical path, as shown by Pierce Jr. and Lala [26]. The delay for m-out-of-2m code checkers also continues to increase with increasing data lengths because the cellular implementation requires “$m (= \text{input data length})$” gate levels in the critical path.

5.2.3. Power Dissipation. In order to evaluate the power dissipation of the three checkers, we used the PowerPlay power analyzer tool. We implemented the Berger [24], m-out-of-2m [26], and SEDC checker using Verilog and synthesized the circuits using Altera’s Quartus II software. We targeted the circuit for a Cyclone II EP2C5A256 A7 chip, which has the least power dissipating properties among the Cyclone family. We allowed the synthesizer to create a balance between area and delay while synthesizing in order to get a better power estimate. We also enabled the synthesizer to use synthesizing model that takes intensive steps to optimize power for all three circuits. We clocked the inputs of the circuit with the default toggle rate and estimated the total thermal power dissipation for different values of input data width.

Table 4: Area overhead of Berger [26], SEDC, and m-out-of-2m [24] code checkers.

<table>
<thead>
<tr>
<th>Data Bit</th>
<th>Berger Code</th>
<th>SEDC</th>
<th>m-out-of-2m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code</td>
<td>Check</td>
<td>Storage</td>
</tr>
<tr>
<td></td>
<td>Code Area</td>
<td>Area</td>
<td>Area</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>24</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>8</td>
<td>112</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
<td>180</td>
<td>228</td>
</tr>
<tr>
<td>5</td>
<td>36</td>
<td>178</td>
<td>230</td>
</tr>
<tr>
<td>7</td>
<td>36</td>
<td>396</td>
<td>456</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>550</td>
<td>626</td>
</tr>
<tr>
<td>15</td>
<td>48</td>
<td>1106</td>
<td>1210</td>
</tr>
<tr>
<td>16</td>
<td>60</td>
<td>1308</td>
<td>1428</td>
</tr>
<tr>
<td>30</td>
<td>60</td>
<td>2586</td>
<td>2762</td>
</tr>
<tr>
<td>32</td>
<td>72</td>
<td>3048</td>
<td>3240</td>
</tr>
</tbody>
</table>

compared to m-out-of-2m code checkers. Although Berger and m-out-of-2m checkers are TSC, while the proposed SEDC checker is only FS, all three checkers provide the same fault security.

5.3. Fault Coverage of the Proposed HW-Level Fault Tolerance Scheme. In order to elaborate the effectiveness of the SEDC CSG and its FS checker, we computed the fault coverage of the proposed SEDC-based HW-level fault tolerance scheme. We applied faults in the example circuit of Figure 4, given in Section 3.4. As most of the VLSI combinational circuits designed for mathematical operations, like add, subtract, multiply, division, etc., consist of multiple instances of 1-bit adders (full adders), hence the example circuit, i.e., a 4-bit adder, is a simple and good candidate for presenting the effectiveness of our scheme. We injected two major types of transient errors, i.e., stuck-at-0 and stuck-at-1 [29], at 24 nodes (at 6 nodes per full adder, as shown in Figure 11(b)). We injected these errors using 2-to-1 multiplexers, whose output is given by

$$m_{\text{mux}} = \begin{cases} \text{false} & \text{if } f_{\text{enable}} = 0 \\ \text{true} & \text{if } f_{\text{enable}} = 1 \end{cases}$$

In Figure 11(a), the symbols $A[3:0]$, $B[3:0]$, $Cin$, $f_{\text{enable}}$, and $F[23:0]$ denote the 4-bits input $A$, 4-bits input $B$, 1-bit carry-in, 1-bit fault enabling signal, and 24-bits fault signals, respectively, while $Cout$ is the carry-out and $S[3:0]$ represents the 4-bits sum output of the 4-bits adder. Figure 11(b) shows the detailed schematic of a single full adder.

We considered that the faults can occur at the outputs of the logic gates only and adopted a single-fault model according to which only one fault can occur at a time [29].
Figure 10: Comparison of (a) power dissipation and (b) area in terms of LE counts, between Berger [26], m-out-of-2m [24], and SEDC checkers.

Figure 11: (a) RTL schematic of a 4-bit adder, and (b) 1-bit full adder, with fault injection.
Table 4: Critical path (CP) delay comparison of Berger, SEDC, and m-out-of-2m codes checker (unit = microseconds).

<table>
<thead>
<tr>
<th>Data Bits</th>
<th>Berger</th>
<th>SEDC</th>
<th>m-out-2m</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.888</td>
<td>0.514</td>
<td>1.024</td>
</tr>
<tr>
<td>3</td>
<td>4.151</td>
<td>2.524</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>7.741</td>
<td>2.738</td>
<td>5.490</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>2.713</td>
<td>5.558</td>
</tr>
<tr>
<td>7</td>
<td>7.821</td>
<td>2.77</td>
<td>8.297</td>
</tr>
<tr>
<td>8</td>
<td>7.599</td>
<td>2.76</td>
<td>9.284</td>
</tr>
<tr>
<td>15</td>
<td>10.566</td>
<td>2.826</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>12.956</td>
<td>2.751</td>
<td>-</td>
</tr>
<tr>
<td>32</td>
<td>17.964</td>
<td>2.771</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5: Summary of fault testing experiment on SEDC-based fault tolerant 4-bit adder.

<table>
<thead>
<tr>
<th>(a) Total errors at the output of the adder</th>
<th>(b) BEs</th>
<th>(c) Detected BEs</th>
<th>(d) UEs</th>
<th>(e) Detected UEs</th>
<th>(f) Total detected errors</th>
<th>(g) Total undetected errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>1748</td>
<td>252</td>
<td>120</td>
<td>1496</td>
<td>1496</td>
<td>1616</td>
</tr>
<tr>
<td>Percentage (%)</td>
<td>100</td>
<td>14.42 w.r.t. (a)</td>
<td>47.62 w.r.t. (b)</td>
<td>85.58 w.r.t. (a)</td>
<td>100 w.r.t. (d)</td>
<td>92.45 w.r.t. (a)</td>
</tr>
</tbody>
</table>

We used Altera’s Quartus II software to design and synthesize the overall system and then simulated the system using ModelSim. We designed a self-checking test bench to evaluate the overall fault coverage. The statistics of the fault injection and its results are summarized in Table 5.

In total, we injected 6425 faults exhaustively, out of which 1748 faults actually caused a logical error at the output of the adder circuitry. Only 14.42% of these injected faults resulted in bidirectional errors (BEs), while most of the faults caused unidirectional errors (UEs). This also proved the fact that most of the errors in VLSI circuits result in UEs at the output [19–21]. Even though SEDC is an AUED scheme, and it provides 100% fault coverage against UEs, it also successfully detected 47.62% of the BEs, as shown in Table 5. This is due to the reason that SEDC partitions the input data word into multiple parts and encodes and decodes each part independently. Consequently, a subset of BEs is also partitioned into multiple UEs and thus detected by the proposed SEDC scheme.

5.4. Cost Analysis: SW-Based Fault Tolerance Versus HW-Based Fault Tolerance. In this section, we discuss the effect of fault propagation and the estimated cost of recovery from failure (also known as repair time) in big data computing in two cases: (a) when HW-based fault tolerance is applied, and (b) when only SW-based fault tolerance is applied. For simplicity in our analysis, we take the example of a coordinated checkpointing (CC) algorithm, which is widely used in HDFS for data recovery [31].

In HDFS, an image is used to define metadata (which contains node data and a list of blocks belonging to each file), while checkpoint defines the persistent record of the image, stored on a secondary NameNode (SNN) (also called DataNode) or Checkpoint Node, or in some cases on the primary NameNode (PNN) itself. If the PNN uses the CC data recovery algorithm, the checkpoints are distributed among multiple SNNs. During normal operation, the SNN sends heartbeats (a communication signal) to the PNN periodically. If the PNN does not receive a heartbeat from the SNN for certain fixed amount of time, the SNN is considered to be out of service, and the block replicas it hosts are considered to be unavailable. In this case, the PNN initiates the CC recovery algorithm, which includes signaling (sending heartbeats with control signals to other nodes) and replicating the copy of failed SNN data (available on the checkpoint nodes) to the other nodes in a coordinated way [31].

For our cost analysis, we would like to compute the cost associated with the CC data recovery algorithm for which we assume a cloud application, such as a message passing interface (MPI) program that comprises \( p \) logical processes that communicate through message passing (heartbeats). Each process is executed on a virtual machine and sends a message to remaining \( p-1 \) processes with equal probabilities. We also consider that the message sending, checkpointing, and fault occurrence events are independent of each other. Assuming that a process is modelled as a sequence of deterministic events, i.e., every step taken by the process has a known outcome, and failure only occurs during message passing with equal probability and not during checkpointing or recovery, we use the analytical cost model given in [4] for cost analysis of fault tolerance at the SW level. According to [4], \( T \) denotes the total execution time of a process without fault tolerance, while \( T_{CP} \) and \( T_{RO} \) represent the checkpointing and failure recovery overheads, respectively. Then, the total cost of fault tolerance per process is given by

\[
C = \frac{T_{CP} + T_{RO}}{T} \times 100
\]
Assuming that the average time to roll back a failed process is $C_{rb}$ and mean time between failures is $1/P(f)$, where $P(f)$ denotes the probability of failure, then according to [4], the average recovery cost in CC per process is given by

$$T_{RO} = \frac{C_{rb}}{1/P(f)} = P(f) C_{rb} \tag{10}$$

Let $P(cp)$ denote the probability that a process starts checkpointing, then $(1 - P(cp))^p$ becomes the probability that $p$ processes do not start checkpointing, while $1 - (1 - P(cp))^p$ becomes the probability that at least one process starts a checkpoint. Consequently, $1/(1 - (1 - P(cp))^p)$ represents the checkpointing interval. A process can be the initiator of checkpointing with probability $1/p$ and generate request (REQ) and acknowledgement signals (ACK) to the rest of the $p - 1$ noninitiators (total $2(p - 1)$ signals) and likewise be a noninitiator with probability $1 - 1/p$ and generate only one ACK signal in response to the initiator. As a result, there are $3(p - 1)/p$ average messages generated per checkpoint, and the average overhead per checkpoint is $C_w + (3(p - 1)/p) C_{nl}$, where $C_w$ denotes the average time to write a checkpoint to a stable node and $C_{nl}$ denotes the average network latency. Then, the average checkpointing cost for a process is given by

$$T_{CP} = C_w + \frac{3(p - 1)/p C_{nl}}{1/(1 - (1 - P(cp))^p)} \tag{11} = \left(1 - (1 - P(cp))^p\right) \left( C_w + \frac{3(p - 1)/p}{p} C_{nl} \right)$$

Using the cost model given in (9), (10), and (11), we carried out the cost of data recovery in the CC algorithm with the parameters, $p = 128$ processes (virtual machines), $P(cp) = 1/15$ (one checkpointing per 15 minutes), $C_{nl} = 20 \text{ msecs}$, $C_w = 1 \text{ sec}$, $C_{rb} = 2 \text{ secs}$, as given in [4]. We consider the value of $P(f) = 1/168$ which implies that 100% of the faults in hardware are propagated to the SW level in the absence of HW-level fault tolerance, while each fault occurs after 168 hours (one week’s time). After we apply HW-level fault tolerance, the probability of failure $P(f)$ reduces to $P'(f) = 0.755 \times P(f)$, where the value 0.755 signifies that only 7.55% of the faults are unhandled by the proposed HW-level fault tolerance system (see Table 5). We vary one of the above parameters by keeping the other constant and observe the effect of data recovery cost with and without the proposed HW-level fault tolerance.

The graph in Figure 12(a) shows the average cost of data recovery when the number of processes $p$ is increased from 32 to 4096 (virtual machines). We consider that an application is partitioned into $p$ processes and each process runs on a virtual machine. The increase in number of processes causes a sharp increase in data recovery cost in the CC algorithm because every process has to coordinate with each other in case of a failure.

Figure 12(b) depicts the effect of network latency on the cost of data recovery. In this case we increased the network latency from 2 milliseconds to 300 milliseconds. Network latency depends heavily upon the traffic situation, network bandwidth, data size, and number of active nodes in the network. Figure 12(b) shows that increasing network latency has a negative impact on data recovery because it takes a longer time for processes to communicate with each other, resulting in delayed data recovery.

Figure 13 illustrates the situation where we increase the checkpointing frequency from one checkpoint per hour (1/60) to one checkpoint per minute. Even though the increase in checkpointing frequency improves the overall fault tolerance, it also increases the overall fault tolerance overhead, as shown in Figure 13.

Finally, we show the effect of the increasing probability of failure on the cost of data recovery in Figure 14. We varied the failure frequency from one failure per 1024 hours to one...
failure per 2 hours, which caused a huge impact on fault
tolerance overhead, as shown in Figure 14. But, if we detect
most of the errors at the hardware level, the average cost
of data recovery reduces to a tolerable limit, as shown in
Figure 14.

Because of the errors arising at the HW level, the average
cost of data recovery in terms of percent increase in runtime
in all of the above cases is much higher if we apply fault
tolerance at the SW level only. Among the four parameters,
i.e., # of processes, network latency, checkpointing frequency,
and frequency of failure, frequency of failure has the worst
effect on the average cost of data recovery. The proposed HW-
level fault tolerance reduces the average cost to a tolerable
limit, which is promising for big data and cloud computing
applications. Although there is a one-time cost associated
with HW-level fault tolerance, it provides high reliability
against potential failures leading to severe socioeconomic
consequences in big data and cloud computing.

6. Conclusions and Future Work

In this paper, we presented a concurrent error detection
coding-based HW-level fault tolerance scheme for big data
and cloud computing. The proposed method uses SEDC
codes to protect against transient errors, which is a major
problem in modern VLSI circuits. We also presented an FS
SEDC checker that not only detects errors in the functional
circuitry but also remains failsafe under s-a-1, s-a-0, s-open,
and s-short errors within checker circuitry. We compared
the performance of the proposed SEDC checker with Berger
and m-out-of-2m checker in terms of area, delay, and power
dissipation, which proves the superiority of the proposed
SEDC checker. Using the example of a 4-bit adder circuit,
we presented a complete SEDC-based HW-level fault toler-
ance system and computed its fault coverage by exhaustive
fault injection. The SEDC-based HW-level fault tolerance
method shows 100%, 47%, and 92.5% fault coverage against
unidirectional, bidirectional, and total errors, respectively.
In order to show the effectiveness of the proposed SEDC-
based HW-level fault tolerance method in big data and cloud
computing applications, we compared the average cost of
fault tolerance overhead with and without HW-level fault
tolerance. The results show that HW-level fault tolerance
reduces the probability of failure due to transient errors,
consequently reducing the average cost of fault tolerance
overhead to a great extent when compared with SW level fault
tolerance only.

From hardware-level evolution such as microprocessors,
memories, and parallel computing devices, to system-level
advancements such as networking, data security, resource
sharing protocols, and operating systems, the underlying technologies have changed a lot since the emergence of big data and cloud computing. Fault tolerance plays a vital role in big data and cloud computing because of the uncertain failures associated with the huge amount of data, both at SW and HW levels. Given this, we believe that this research opens new opportunities for fault tolerance at the hardware-level for big data and cloud computing.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was partly supported by research funds from Chosun University, 2017, Sogang University Research Grant of 2012 (201210056.01) and MISP (Ministry of Science, ICT & Future Planning), Korea, under the National Program for Excellence in SW (2015-0-00910) supervised by the IITP (Institute for Information & communications Technology Promotion).

References


Research Article

Research on the Prewarning Method for the Safety of South-to-North Water Transfer Project Driven by Monitoring Data

Yang Liu, Yaoling Fan, Xinqing Yan, Xuemei Liu, and Bin Yang

School of Information Engineering, North China University of Water Resources and Electric Power, Zhengzhou 450011, China

Correspondence should be addressed to Yang Liu; yangliu@ncwu.edu.cn

Received 8 November 2017; Accepted 4 March 2018; Published 4 June 2018

Academic Editor: Shangguang Wang

Copyright © 2018 Yang Liu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In order to solve the prewarning problem of South-to-North Water Transfer Project safety, an intelligent cooperative prewarning method based on machine learning was proposed under the framework of intelligent information processing. Driven by the monitoring data of the South-to-North Water Transfer Project, the single sensor in typical scenes was studied, and the security threshold was predicted along the vertical axis of time, firstly. With the support of the data correlation calculation, the sensors in the typical scene were intelligently grouped, and the study objectives were changed into sensor grouping, secondly. Then, the nonlinear regression model between the single sensor and the multisensors was built on the time cross section, and the model was used to dynamically calculate the safety threshold of the current sensor for the second time. Finally, in the framework of intelligent information processing, a double verification mechanism was proposed to support the construction of the intelligent prewarning method for the safety of South-to-North Water Transfer Project. The paper collected the monitoring data from November 2015 to September 2016 in the typical scenarios. The experimental results showed that the methods constructed in the paper can be able to identify the abnormal causes of data sudden jump effectively and give the different level prewarning. The method provides a strong theoretical support for further manual investigation work.

1. Research Background

The middle route of the South-to-North Water Transfer Project diverted water from the Dan Jiang Kou reservoir to Henan Province, Hebei Province, Tianjin, and Beijing. The project length was 1432 km. The complexity of the geological and meteorological conditions along the project made the South-to-North Water Transfer Project face serious challenges. The engineering safety referred to the safety problems of the middle route of the South-to-North Water Transfer Project mainly including the buildings, channels, and the important engineering facilities. In the actual monitoring process, the number of sensors was very large; at the same time, a lot of sensors were often installed in the bottom of the canal or were embedded in the channel projects, so the routine maintenance and maintenance work for these sensors was very difficult to implement. When the monitoring data was abnormal, we cannot judge whether the data anomaly was caused by sensors or by the channel engineering failure; thus, the staff could not grasp the overall situation of the channel security. In view of this problem, under the driving of the safety monitoring data of the South-to-North Water Transfer Project, the paper regarded a single sensor as research object and predicted the safety threshold along the time axis by Kalman filter method based on its historical monitoring data, firstly; then, the paper expanded the research object from single sensor to sensor grouping by using the data correlation analysis methods, which can reduce the computational complexity and improve the accuracy of prediction algorithm. Secondly, the nonlinear regression model between single sensor and sensor grouping was built to predict and check the sensor’s data on the time cross section. Finally, the intelligent prewarning method was constructed under the framework of intelligent information processing, which provided scientific theoretical support and effective decision-making for the emergency troubleshooting and emergency countermeasures.
The first part of paper introduced the research background. The second part introduced the basic principle of several typical machine learning methods. The third part introduced the data prediction based on machine learning methods. The fourth part introduced the basic principle and processing flow of intelligent prewarning method. In the fifth part, the algorithm was validated and the results were analyzed.

2. Principles of Machine Learning Methods

2.1. The Basic Principles of Random Forest (RF). The Random Forest (RF) algorithm was a kind of machine learning model proposed by Leo Breiman in 2001 [1, 2]. The RF method generated a lot of classification trees by randomly using attributes (columns) and data (rows) of the sample set and finally summarizes these classification trees to form the final Random Forest. Each tree in a RF is a binary tree, and its generation followed the top-down recursive splitting principle; in other words, the training set is divided from the root node in turn. In the binary tree, the root node contained all the training data and was divided into left node and right node which contained a subset of training data, in accordance with the principle of minimum node purity. Then, the left and right nodes continued to be split according to the same rule, until the stop rule was satisfied. If the classification data on one node n all came from the same class, then the purity of this node was \( I(n) = 0 \). The specific implementation process of RF was as follows:

(i) The original training set is \( N \), RF method randomly extracted \( k \) new samples and constructed \( k \) classification trees by using the bootstrap method; each time the samples which were not extracted would form the out of pocket data set.

(ii) Suppose there were \( m_{all} \) variables. \( m_{try} \) variables were randomly extracted at each node of the classification tree, and the most powerful variable is selected in the \( m_{try} \). The threshold of variable classification is determined by checking each classification point.

(iii) Each tree would grow for the maximum without any pruning.

(iv) Finally, Random Forest was composed of the decision trees which were produced by (a), and the new data was discriminated and classified by the Random Forest classifier. The classification result depended on the number of votes of the tree classifier.

2.2. The Basic Principles of Ada-Boost. Ada-Boost was an acronym for “Adaptive Boosting” in English presented by Robert Schapire in 1995 [3, 4]. Its adaptation was that the sample which was classified inaccurately by the previous basic classifier would be strengthened. Then, the whole sample would be used again to train the next basic classifier. At the same time, a new weak classifier would be added to the classifier set each circle until a predetermined small error rate was reached or the maximum number of iterations specified in advance was reached.

Ada-Boost algorithm implementation process was as follows:

1. Initialize the weight of each training sample; the number of the total training examples was \( N \). The calculation of initialization was as formula (1):

\[
W = (w_{11}, w_{12}, \ldots , w_{1N}) , \quad w_{i i} = \frac{1}{N} \quad i = 1, 2, \ldots , N \quad (1)
\]

2. The algorithm was trained in \( M \) circles; the \( m \) circle of learning process was as follows:

(a) Use the training samples with weight distribution \( W_m \) to get the base classifier \( G_m \).

(b) Calculate the error rate of the base classifier obtained in the previous step:

\[
e_m = P (G_m (x_i) \neq y_i) = \frac{\sum_{i=1}^{N} w_{mi} I (G_m (x_i) \neq y_i)}{\sum_{i=1}^{N} w_{mi}} \quad (2)
\]

(c) Calculate the weighting factor in front of \( G_m \):

\[
\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} \quad (3)
\]

(d) Updated the weight coefficient of the training sample:

\[
W_{m+1,j} = \frac{w_{mi}}{Z_m} \exp (-\alpha_m y_i G_m (x_i)) \quad (4)
\]

\[
Z_m = \sum_{i=1}^{N} w_{mi} \exp (-\alpha_m y_i G_m (x_i))
\]

(e) Repeat (a) to (d) to obtain a series of weight parameters \( \alpha_m \) and the base classifier \( G_m \).

3. The base classifier obtained in the previous step would be combined linearly according to the weight parameters to obtain the final classifier:

\[
f (x) = \sum_{m=1}^{M} \alpha_m G_m (x)
\]

\[
G (x) = \text{sign} (f (x)) = \text{sign} \left( \sum_{i=1}^{N} \alpha_m G_m (x) \right) \quad (5)
\]

There were four machine learning algorithms in the third section of the paper. In addition to the Ada-Boost method and Random Forest method, the Bagging algorithm and the support vector machine (SVM) method were used as the contrast method in the paper. Because of the limitation of space, they were no longer detailed here. The basic principles and applications of the methods could be seen in the literature [5-7].
2.3. The Basic Principles of SVR. SVR is a machine learning method which can be used for time series prediction. Through a nonlinear kernel function, the multidimensional input is mapped onto the feature space of higher dimension and then the regression operation is performed to obtain the nonlinear mapping relation with the output index. Due to space limitations, the implementation details of SVR method are shown in the literature [8].

3. The Data Prediction Based on the Machine Learning

3.1. K-Fold Cross-Validation. Cross-validation was a statistical analysis method to verify the performance of the algorithm. The basic idea of cross-validation is that the raw data was grouped into two subsets in some sense, one of the subset was used as the training set (train set), and the other subset was used as the validation set (validation set). The training set is used to train the model, and then the validation set is used to test the performance of the model which was obtained by the first step.

K-fold cross-validation is one of the most commonly used methods for data validation in cross-validation. The original data was divided into K groups (generally equal); each subset of data was a verification set, and the rest of the K – 1 subset data as training set. The process of K-fold cross-validation was as follows.

Step 1. The whole sample set S was divided into k disjoint subsets, assuming that the number of samples in S was m; then each subset has m/k training samples, and the corresponding subset is called S’[s1, s2, ..., sk].

Step 2. Each subset in S’ would be picked out as the test set, and the other k – 1 as the training set.

Step 3. Obtain the model or hypothesis according to the training set.

Step 4. The training model was used to classify on the test set, and the accuracy of classification would be calculated.

Step 5. The mean value of the correct classification rate calculated by K times was used as the true classification rate of the model or the assumed function.

3.2. Data Prediction under the 6-Fold Cross-Validation. When the monitoring data of current sensor was abnormal, the method would calculate the number of the sensors whose monitoring data was abnormal at that moment. The cooperative prewarning algorithm would send out an engineering red warning that mean the project was danger, if the number of sensors in the same group of sensors is higher than 60%. If the ratio was less than 60%, a nonlinear regression model between the current sensor and the residual sensors within the same group would be established, and the model was used to predict the current sensor monitoring data on the time cross section for the second time. The paper used the 6-fold cross-validation method, 16% of the total samples were randomly selected as the test samples, and the remaining 84% samples were used as the training samples. The machine learning algorithms investigated in the paper included Bagging, SVM, Ada-Boost, and Random Forest. Because of the limited space, only the prediction result of sensor R1_4 is shown here. In the prediction process, Bagging algorithm, Ada-Boosting algorithm, and Random Forest method all use the regression trees as basic modes, and the number of regression trees was 50, and the depth is 30. The constant C = 10⁻², gamma = 1 in SVR algorithm.

Figure 1 showed the prediction result curves of the sensor R1_4 under the four machine learning methods. The overall trend of monitoring data of sensor R1_4 was relatively stable, as we can see that the Ada-Boosting, SVR, and Random Forest methods showed good tracking performance. When the monitoring data had abrupt jump, the prediction curve of Ada-Boost method showed good convergence and can track the data jump in time. At the same time, the prediction result of SVR method showed a large fluctuation, and the prediction curve had obvious deviation. From the data prediction curve, we can see that the SVR algorithm is too sensitive to the data fluctuation. Figure 6 showed the prediction error curve of sensor R1_4 under various methods. From the error curve, we can see that, in the prediction process of sensor R1_4, the SVR method has good prediction accuracy when the data is stationary. When the data fluctuates slightly, the SVR prediction produces a larger prediction deviation. It can be seen from the whole prediction process that the Ada-Boost method has better performance in prediction accuracy and algorithm stability. Figure 2 showed the lines of mean value of the prediction error belonging to every method. We can see that the error line of SVR emerged sudden jump when the monitoring data of the sensor R1_4 has bigger change.

The prediction error of each algorithm was shown in Table 1. It can be seen from the mean value of error that the data prediction accuracy of Ada-Boost algorithm was best under the typical scene, and the worst accuracy was the Bagging algorithm. The statistic results of data prediction were consistent with the curve results; they had the same conclusion. At the same time, the error variance and standard deviation of each algorithm showed that the error variance and standard deviation of Ada-Boost method were the smallest. The Ada-Boost method and Random Forest method had better prediction accuracy and prediction stability in the data prediction. Therefore, they were used to construct the intelligent prewarning method for the South-to-North Water Transfer Project.

4. Design of the Intelligent Prewarning Method

This section focuses on how to construct the time-space cooperative intelligent prewarning information processing method based on the use of machine learning algorithms. When the monitoring data was abnormal, the method would find it in time and judge abnormal course whether the abnormal data was caused by the sensor fault or by the engineering itself faults. Then, the method could send out the different levels prewarning message according to the abnormal course.
The third part of the paper is the basis of the fourth part. The intelligent prewarning method uses the prediction results belonging to the third part; at the same time, it can generate the security range dynamically. This is a standard to judge whether the data is abnormal.

In order to solve the correct prewarning of abnormal data, the paper adopted time-space cooperative verification method based on machine learning methods to forecast and verify the monitoring data of the sensor from time and space dimension, respectively. Generally, the sensor monitoring data changed in linear mode in a relatively short period of time. Here, the paper used the traditional Kalman filter method [8–10] to predict the single-sensor monitoring data on historical monitoring data and, at the same time, generate the security interval of monitoring data based on the prediction results. The whole processing process is shown in Figure 3.

**Step 1.** Based on the Kalman filter, the current monitoring data was predicted by using the historical data of a single sensor in the given time slice. Then the time domain security range was generated by using the predicted data points. If the current monitoring data was in the safe range, we would regard it as a normal data and record it down; otherwise, the method would jump to Step 2.

**Step 2.** Got the sensor group where the current sensor was located.

**Step 3.** Calculate the number of sensors that occur abnormally in the current sensor group. If the abnormal ratio of sensor grouping was greater than the predecision threshold, a high level red prewarning message would be carried out; otherwise, jump to Step 4.

**Step 4.** The nonlinear regression model between the current sensor and the other sensors in the sensor grouping was built based on machine learning methods, and the model was used to predict and check the current monitoring data of the sensor.

### Table 1: The error rate of SVM, Forest, Ada-Boost, and Bagging algorithms in the typical scene.

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Error Mean</th>
<th>Error Variance</th>
<th>Error (max)</th>
<th>Error (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>0.212</td>
<td>11.16</td>
<td>9.474</td>
<td>−6.046</td>
</tr>
<tr>
<td>Forest</td>
<td>−0.006</td>
<td>6.81</td>
<td>6.739</td>
<td>−6.381</td>
</tr>
<tr>
<td>Ada-Boost</td>
<td>−0.003</td>
<td>1.77</td>
<td>4.551</td>
<td>−4.990</td>
</tr>
<tr>
<td>Bagging</td>
<td>0.626</td>
<td>14.02</td>
<td>10.198</td>
<td>−9.229</td>
</tr>
</tbody>
</table>
**Step 5.** The security range of the data was generated on Step 4 prediction results. If the monitoring value of the current sensor was in the new security range, even if the data exceeds the security range produced by Kalman filter in Step 1, it would be treated as normal data. Otherwise, the sensor level alarm would be send out, which indicted the current sensor was wrong. It was suggested that the monitoring data of this equipment should be ignored; otherwise, the overall judgment of the subsequent safety of the project would be affected.

As introduced in “Step 2”, the sensor group was the result based on the Pearson correlation coefficient. Pearson correlation coefficient is a linear correlation coefficient, which is used to reflect the linear correlation of two variables. The correlation coefficient is denoted by r, and it is a value between 1 and −1, where 1 indicates that the variable is completely positive, 0 is independent, and −1 means completely negative correlation. Pearson correlation coefficients are calculated as shown in formula (6).

$$
\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E((X - u_X)(Y - u_Y))}{\sigma_X \sigma_Y} = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - E(X)^2} \sqrt{E(Y^2) - E(Y)^2}} \tag{6}
$$

With the sensors R1_4, R1_7, R1_8, R1_16, R1_18, and R1_19, the correlation coefficients between the two sensors were calculated. Table 2 shows the real-time grouping results of sensor sets. In the table, the sensor group with strong correlation with R1_4 contains {R1_7, R1_18, R1_19}, and the order in the sensor group is arranged according to its correlation with R1_4.

### Table 2: The grouping results of sensor sets.

<table>
<thead>
<tr>
<th>Current Sensor</th>
<th>Sensor Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1_4</td>
<td>R1_7, R1_18, R1_19</td>
</tr>
<tr>
<td>R1_5</td>
<td>R1_16, R1_18, R1_7</td>
</tr>
<tr>
<td>R1_7</td>
<td>R1_18, R1_4, R1_19</td>
</tr>
<tr>
<td>R1_8</td>
<td>R1_19, R1_4, R1_18</td>
</tr>
<tr>
<td>R1_14</td>
<td>R1_8, R1_19, R1_4</td>
</tr>
<tr>
<td>R1_16</td>
<td>R1_5, R1_18, R1_7</td>
</tr>
<tr>
<td>R1_18</td>
<td>R1_7, R1_4, R1_19</td>
</tr>
<tr>
<td>R1_19</td>
<td>R1_4, R1_7, R1_18</td>
</tr>
</tbody>
</table>

**5. Experiment and Analysis**

#### 5.1. The Description of Typical Scene.

The safety monitoring data of channel is from the South-to-North Water Transfer Project Construction Administration. A typical scene of the South-to-North Water Transfer Project was showed in Figure 4, in which a number of steel bars (R1_11 ~ R1_22) were regularly embedded in the project. Their concrete position...
is shown in Figure 4, and the safety parameters of the canal engineering are monitored in real-time. In the process of research, the monitoring data were collected from October 2014 to October 2015 for about one year. The original monitoring data of the steel bars in the typical scene was showed in Figure 5. Take the monitoring data of R1,4 as an example. There are 335 batches data in all, and the average value of temperature is \(-13.06\) degrees centigrade. The mean value of stress is 23.90 MPa, the variance of stress is 3.51, the minimum value is 17.11 MPa, the maximum value is 29.45 MPa, the median value is 23.76 MPa, and the quartiles value is 20.43 MPa.

The computer is configured as follows: processor is Intel core CPU i5-6500, CPU frequency is 3.20 GHz; memory is 4.00 GB; the operating system is Windows 10 (64-bit); the programming language is Python 3.5.2 (64 bits); the integrated development environment is Pycharm Community Edition.

5.2. Realization of Cooperative Intelligent Prewarning Method. Figure 6 showed the prewarning results of steel bars such as R1,4, R1,5, R1,7, R1,8. The prewarning results were produced by the intelligent cooperative prewarning method for the safety of South-to-North Water Transfer Project, which was constructed based on the Ada-Boost method and the Random Forest method. In the figure, the horizontal axis was the monitoring time, and the longitudinal axis was the monitoring data of the steel bars from October 2014 to October 2015. At the same time, the prewarning points of the channel engineering safety produced by intelligent cooperative prewarning method also were marked. The dark blue “*” was the project level prewarning points. Project level warning points mean that the channel project itself may have security risks. It was necessary to organize relevant persons go to the scene immediately for further security investigation. Green “*” was the sensor level warning points based on the Random Forest method, and the yellow “*” was the sensor level warning point based on the Ada-Boost method. These sensor level warning points mean that the data abnormal was caused by the sensor failure and had nothing to do with the channel engineering. In order to know the overall situation of the channel safety accurately in the follow-up work, the data of current sensor could be ignored temporarily, so as to reduce the interference to the results of the intelligent prewarning data processing.

6. Conclusion

Along the Middle Route Project of the South-to-North Water Transfer Project, the geological conditions were complicated, the number of sensors was large, and the position was special. Therefore, the regular maintenance and routine maintenance of these sensors were difficult. So, when the monitoring data was abnormal, it was impossible to judge whether the abnormal data was caused by sensor's fault or by the quality of the channel engineering. To solve this problem, the paper jumped out of the traditional theory research of water conservancy engineering and built the intelligent prewarning method, just from the perspective of data research. At the same time, the method was used in the typical scenario of South-to-North Water Transfer Project.

In this method, the data correlation analysis method was applied to realize the dynamic grouping of sensors. Then, a kind of double checking mechanism of abnormal data was constructed, which could check the monitoring data from the axis of time and on the time cross section. Finally, in the framework of intelligent information processing, data analysis and machine learning methods were organically combined to establish a complete data processing process and engineering safety warning mechanism. The experimental results showed that this method was a feasible and effective method for the abnormal data processing in the South-to-North Water Transfer Project.

Disclosure

This article is original for the first author. It does not contain any conflict with any other article or project (project number: 51509090, project name: The Discovery and Inversion of Emergent Groundwater Contaminant Based on Sequence Mining and Intelligent Computing; project number: 16HASTIT034; project name: Research on Intelligent Computing Method for Large-Scale Space Temporal Sequence Data).

Conflicts of Interest

The authors declare no conflicts of interest related to this work.

Acknowledgments

The research of this paper is partly sponsored by the National Science Foundation of China (under Grants nos. 51509090 and U1604152) and Program for Scientific and Technological Innovation Talents of Colleges and Universities in Henan (under Grant no. 16HASTIT034).
Scientific Programming

Figure 6: The results of intelligent prewarning method based on the RF and Ada-Boost method.

References


Review Article
Big Data in Cloud Computing: A Resource Management Perspective

Saeed Ullah, M. Daud Awan, and M. Sikander Hayat Khiyal

Faculty of Computer Science, Preston University, Islamabad, Pakistan

Correspondence should be addressed to Saeed Ullah; saeedullah@gmail.com

Received 27 October 2017; Revised 11 February 2018; Accepted 25 March 2018; Published 9 May 2018

A cademic Editor: Sungyong Park

Copyright © 2018 Saeed Ullah et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The modern day advancement is increasingly digitizing our lives which has led to a rapid growth of data. Such multidimensional datasets are precious due to the potential of unearthing new knowledge and developing decision-making insights from them. Analyzing this huge amount of data from multiple sources can help organizations to plan for the future and anticipate changing market trends and customer requirements. While the Hadoop framework is a popular platform for processing larger datasets, there are a number of other computing infrastructures, available to use in various application domains. The primary focus of the study is how to classify major big data resource management systems in the context of cloud computing environment. We identify some key features which characterize big data frameworks as well as their associated challenges and issues. We use various evaluation metrics from different aspects to identify usage scenarios of these platforms. The study came up with some interesting findings which are in contradiction with the available literature on the Internet.

1. Introduction

We live in the information age, and an important measurement of present times is the amount of data that is generated anywhere around us. Data is becoming increasingly valuable. Enterprises are aiming at unlocking data’s hidden potential and deliver competitive advantage [1]. Statistics MRC projected that the data analytics and Hadoop market, which accounted for $8.48 billion in 2015, is expected to reach at $99.31 billion by 2022 [2]. The global big data market has estimated that it will jump from $14.87 billion in 2013 to $46.34 billion in 2018 [3]. Gartner has predicted that data will grow by 800 percent over the next five years and 80 percent of the data will be unstructured (e-mails, documents, audio, video, and social media content) and 20 percent will be structured (e-commerce transactions and contact information) [1].

Today’s largest scientific institution, CERN, produces over 200 PB of data per year in the Large Hadron Collider project (as of 2017). The amount of generated data on the Internet has already exceeded 2.5 exabytes per day. Within one minute, 400 hours of videos are uploaded on YouTube, 3.6 million Google searches are conducted worldwide each minute of every day, more than 656 million tweets are shared on Twitter, and more than 6.5 million pictures are shared on Instagram each day. When a dataset becomes so large that its storage and processing become challenging due to the constraints of existing tools and resources, the dataset is referred to as big data [4, 5]. It is the first part of the journey towards delivering decision-making insights. But instead of focusing on people, this process utilizes a much more powerful and evolving technology, given the latest breakthroughs in this field, to quickly analyze huge streams of data, from a variety of sources, and to produce one single stream of useful knowledge [6].

Big data applications might be viewed as the advancement of parallel computing, but with the important exception of the scale. The scale is the necessity arising from the nature of the target issues: data dimensions largely exceed conventional storage units, the level of parallelism needed to perform computation within a strict deadline is high, and obtaining final results requires the aggregation of large numbers of partial results. The scale factor, in this case, does not only have the same effect that it has in classical parallel computing, but
it surges towards a dimension in which automated resource management and its exploitation are of significant value [7].

An important factor for the success in big data analytical projects is the management of resources: these platforms use a substantial amount of virtualized hardware resources to optimize the tradeoff between costs and results. Managing such resources is definitely a challenge. Complexity is rooted in their architecture: the first level of complexity stems from their performance requirements of computing nodes: typical big data applications utilize massively parallel computing resources, storage subsystems, and networking infrastructure because of the fact that results are required within a certain time frame, or they can lose their value over time. Heterogeneity is a technological need: evolvability, extensibility, and maintainability of the hardware layer imply that the system will be partially integrated, replaced, or extended by means of new parts, according to the availability on the market and the evolution of technology [7]. Another important consideration of modern applications is the massive amount of data that need to be processed. Such data usually originate from different sets of devices (e.g., public web, business applications, satellites, or sensors) and procedures (e.g., case studies, observational studies, or simulations). Therefore, it is imperative to develop computational architectures with even better performance to support current and future application needs. Historically, this need for computational resources was provided by high-performance computing (HPC) environments such as computer clusters, supercomputers, and grids. In traditional owner-centric HPC environments, internal resources are handled by a single administrative domain [19]. Cluster computing is the leading architecture for this environment. In distributed HPC environments, such as grid computing, virtual organizations manage the provisioning of resources, both internal and external, to meet application needs [20]. However, the paradigm shift towards cloud computing has been widely discussed in more recent researches [19, 21], targeting the execution of HPC workloads on cloud computing environments. Although organizations usually prefer to store their most sensitive data internally (on-premises), huge volumes of big data (owned by the enterprises or generated by third parties) may be stored externally; some of it may already be on a cloud. Retaining all data sources behind the firewall may result in a significant waste of resources. Analyzing the data where it resides either internally or in a public cloud data center makes more sense [1, 22].

Even if cloud computing has to be an enabler to the growth of big data applications, common cloud computing solutions are rather different from big data applications. Typically, cloud computing solutions offer fine-grained, loosely coupled applications, run to serve large numbers of users that operate independently, from multiple locations, possibly on own, private, nonshared data, with a significant amount of interactions, rather than being mainly batch-oriented, and generally fit to be relocated with highly dynamic resource needs. Despite such differences, cloud computing and big data architectures share a number of common requirements, such as automated (or autonomic) fine-grained resource management and scaling related issues [7].

As cloud computing begins to mature, a large number of enterprises are building efficient and agile cloud environments, and cloud providers continue to expand service offerings [1]. Microsoft’s cloud Hadoop offering includes Azure Marketplace, which runs Cloudera Enterprise, MapR, and Hortonworks Data Platform (HDP) in a virtual machine, and Azure Data Lake, which includes Azure HDInsight, Data Lake Analytics, and Data Lake Store as managed services. The platform offers rich productivity suites for database, data warehouse, cloud, spreadsheet, collaboration, business intelligence, OLAP, and development tools, delivering a growing Hadoop stack to Microsoft community. Amazon Web Services reigns among the leaders of cloud computing and big data solutions. Amazon EMR is available across 14 regions worldwide. AWS offers versions of Hadoop, Spark, Tez, and Presto that can work off data stored in Amazon S3 and Amazon Glacier. Cloud Dataproc is Google’s managed Hadoop and Spark cluster to use fully managed cloud services such as Google BigQuery and Bigtable. IBM differentiates BigInsights with end-to-end advanced analytics. IBM BigInsights runs on top of IBM’s SoftLayer cloud infrastructure and can be deployed on more than 30 global data centers. IBM is making significant investments in Spark, BigQuality, BigIntegrate, and IBM InfoSphere Big Match that run natively with YARN to handle the toughest Hadoop use cases [23].

In this paper, we give an overview of some of the most popular and widely used big data frameworks, in the context of cloud computing environment, which are designed to cope with the above-mentioned resource management and scaling problems. The primary object of the study is how to classify different big data resource management systems. We use various evaluation metrics for popular big data frameworks from different aspects. We also identify some key features which characterize big data frameworks as well as their associated challenges and issues. We restricted our study selection criteria to empirical studies from existing literature with reported evidence on performance evaluation of big data resource management frameworks. To the best of our knowledge, thus far there has been no empirical based performance evaluation report on major resource management frameworks. We investigated the validity of existing research by performing a confirmatory study. For this purpose, the standard performance evaluation tests as well as custom load test cases were performed on a 10+1 nodes t2.2xlarge Amazon AWS cluster. For experimentation and benchmarking, we followed the same process as outlined in our earlier study [24].

The study came up with some interesting findings which are in contradiction with the available literature on the Internet. The novelty of the study includes the categorization of cloud-based big data resource management frameworks according to their key features, comparative evaluation of the popular big data frameworks, and the best practices related to the use of big data frameworks in the cloud.

The inclusion and exclusion criteria for relevant research studies are as follows:

(i) We selected only those resource management frameworks for which we found empirical evidence of being offered by various cloud providers.
(ii) Several vendors offer their proprietary solutions for big data analysis which could be the potential candidate for comparative analysis being conducted in this study. However, these frameworks were not selected based on two reasons. Firstly, most of these solutions are the extension of open-source solution and hence these exhibit the identical perform results in most of the cases. Secondly, for our empirical studies, researchers mostly prefer open-source solutions as the documentation, usage scenarios, source code, and other relevant details are freely available. Hence, we selected open-source solutions for the performance evaluation.

(iii) We did not include the frameworks which are now deprecated or discounted, such as Apache S4, in favor of other resource management systems.

This paper is organized as follows. Section 2 reviews the popular resource management frameworks. The comparison of big data frameworks is presented in Section 3. Based on the comparative evaluation, we categorize these systems in Section 4. Related work is presented in Section 5 and, finally, we present conclusion and possible future directions in Section 6.

2. Big Data Resource Management Frameworks

Big data is offering new emerging trends and opportunities to unearth operational insight towards data management. The most challenging issues for organizations are often that the amount of data is massive which needs to be processed at an optimal speed to synthesize relevant results. Analyzing such huge amount of data from multiple sources can help organizations plan for the future and anticipate changing market trends and customer requirements. In many of the cases, big data is analyzed in batch mode. However, in many situations, we may need to react to the current state of data or analyze the data that is in motion (data that is constantly coming in and needs to be processed immediately). These applications require a continuous stream of often unstructured data to be processed. Therefore, data is continuously analyzed and cached in memory before it is stored on secondary storage devices. Processing streams of data works by filtering in-memory tables of data across a cluster of servers. Any delay in the data analysis can seriously impact customer satisfaction or may result in project failure [25].

While the Hadoop framework is a popular platform for processing huge datasets in parallel batch mode using commodity computational resources, there are a number of other computing infrastructures that can be used in various application domains. The primary focus of this study is to investigate popular big data resource management frameworks which are commonly used in cloud computing environment. Most of the popular big data tools available for cloud computing platform, including the Hadoop ecosystem, are available under open-source licenses. One of the key appeals of Hadoop and other open-source solutions is the low total cost of ownership. While proprietary solutions have expensive license fees and may require more costly specialized hardware, these open-source solutions have no licensing fees and can run on industry-standard hardware [14]. Figure 1 demonstrates the classification of various styles of processing architectures of open-source big data resource management frameworks.

In the subsequent section, we discuss various open-source big data resource management frameworks that are
widely used in conjunction with cloud computing environment.

2.1. Hadoop. Hadoop [26] is a distributed programming and storage infrastructure based on the open-source implementation of the MapReduce model [27]. MapReduce is the first and current de facto programming environment for developing data-centric parallel applications for parsing and processing large datasets. The MapReduce is inspired by Map and Reduce primitives used in functional programming. In MapReduce programming, users only have to write the logic of Mapper and Reducer while the process of shuffling, partitioning, and sorting is automatically handled by the execution engine [14, 27, 28]. The data can either be saved in the Hadoop file system as unstructured data or in a database as structured data [14]. Hadoop Distributed File System (HDFS) is responsible for breaking large data files into smaller pieces known as blocks. The blocks are placed on different data nodes, and it is the job of the NameNode to notice what blocks on which data nodes make up the complete file. The NameNode also works as a traffic cop, handling all access to the files, including reads, writes, creates, deletes, and replication of data blocks on the data nodes. A pipeline is a link between multiple data nodes that exists to handle the transfer of data across the servers. A user application pushes a block to the first data node in the pipeline. The data node takes over and forwards the block to the next node in the pipeline; this continues until all the data, and all the data replicas, are saved to disk. Afterwards, the client repeats the process by writing the next block in the file [25].

The two major components of Hadoop MapReduce are job scheduling and tracking. The early versions of Hadoop supported limited job and task tracking system. In particular, the earlier scheduler could not manage non-MapReduce tasks and it was not capable of optimizing cluster utilization. So, a new capability was aimed at addressing these shortcomings which may offer more flexibility, scaling, efficiency, and performance. Because of these issues, Hadoop 2.0 was introduced. Alongside earlier HDFS, resource management, and MapReduce model, it introduced a new resource management layer called Yet Another Resource Negotiator (YARN) that takes care of better resource utilization [25].

YARN is the core Hadoop service to provide two major functionalities: global resource management (ResourceManager) and per-application management (ApplicationMaster). The ResourceManager is a master service which controls NodeManager in each of the nodes of a Hadoop cluster. It includes a scheduler, whose main task is to allocate system resources to specific running applications. All the required system information is tracked by a Resource Container which monitors CPU, storage, network, and other important resource attributes necessary for executing applications in the cluster. The ResourceManager has a slave NodeManager service to monitor application usage statistics. Each deployed application is handled by a corresponding ApplicationMaster service. If more resources are required to support the running application, the ApplicationMaster requests the NodeManager and the NodeManager negotiates with the ResourceManager (scheduler) for the additional capacity on behalf of the application [26].

2.2. Spark. Apache Spark [29], originally developed as Berkeley Spark, was proposed as an alternative to Hadoop. It can perform faster parallel computing operations by using in-memory primitives. A job can load data in either local memory or a cluster-wide shared memory and query it iteratively with much greater speed as compared to disk-based systems such as Hadoop MapReduce [27]. Spark has been developed for two applications where keeping data in memory may significantly improve performance: iterative machine learning algorithms and interactive data mining. Spark is also intended to unify the current processing stack, where batch processing is performed using MapReduce, interactive queries are performed using HBase, and the processing of streams for real-time analytics is performed using other frameworks such as Twitter's Storm. Spark offers programmers a functional programming paradigm with data-centric programming interfaces built on top of a new data model called Resilient Distributed Dataset (RDD) which is a collection of objects spread across a cluster stored in memory or disk [28]. Applications in Spark can load these RDDs into the memory of a cluster of nodes and let the Spark engine automatically manage the partitioning of the data and its locality during runtime. This versatile iterative model makes it possible to control the persistence and manage the partitioning of data. A stream of incoming data can be partitioned into a series of batches and is processed as a sequence of small-batch jobs. The Spark framework allows this seamless combination of streaming and batch processing in a unified system. To provide rapid application development, Spark provides clean, concise APIs in Scala, Java, and Python. Spark can be used interactively from the Scala and Python shells to rapidly query big datasets.

Spark is also the engine behind Shark, a complete Apache Hive-compatible data warehousing system that can run much faster than Hive. Spark also supports data access from Hadoop. Spark fits in seamlessly with the Hadoop 2.0 ecosystem (Figure 2) as an alternative to MapReduce, while using the same underlying infrastructure such as YARN and the HDFS. Spark is also an integral part of the SMACK stack to provide the most popular cloud-native PaaS such as IoT, predictive analytics, and real-time personalization for big data. In SMACK, Apache Mesos cluster manager (instead of YARN) is used for dynamic allocation of cluster resources, not only for running Hadoop applications but also for handling heterogeneous workloads.

The GraphX and MLlib libraries include state-of-the-art graph and machine learning algorithms that can be executed in real time. BlinkDB is a novel parallel, sampling-based approximate query engine for running interactive SQL queries that trade off query accuracy for response time, with results annotated by meaningful error bars. BlinkDB has been proven to run 200 times faster than Hive within an error rate of 2–10%. Moreover, Spark provides an interactive tool called Spark Shell which allows exploiting the Spark cluster in real time. Once interactive applications are created, they may subsequently be executed interactively in the cluster.
Figure 2: Hadoop 2.0 ecosystem, source: [14].

In Figure 3, we present the general Spark system architecture.

2.3. Flink. Apache Flink is an emerging competitor of Spark which offers functional programming interfaces, much similar to Spark. It shares many programming primitives and transformations in the same way as what Spark does for iterative development, predictive analysis, and graph stream processing. Flink is developed to fill the gap left by Spark, which uses minibatch streaming processing instead of a pure streaming approach. Flink ensures high processing performance when dealing with complex big data structures such as graphs. Flink programs are regular applications which are written with a rich set of transformation operations (such as mapping, filtering, grouping, aggregating, and joining) to the input datasets. The Flink dataset uses a table-based model; therefore application developers can use index numbers to specify a particular field of a dataset [27, 28].

Flink is designed to run on large-scale clusters with thousands of nodes, and in addition to a standalone cluster mode, Flink provides support for YARN. For distributed environment, Flink chains operator subtasks together into tasks. Each task is executed by one thread [16]. Flink runtime consists of two types of processes: there is at least one JobManager (also called masters) which coordinates the distributed execution. It schedules tasks, coordinates checkpoints, and coordinates recovery on failures. A high-availability setup may involve multiple JobManagers, one of which is always the leader, and the others are standby. The TaskManagers (also called workers) execute the tasks (or, more specifically, the subtasks) of a dataflow/buffer and exchange the data streams. There must always be at least one TaskManager. The JobManagers and TaskManagers can be started in various ways: directly on the machines as a standalone cluster, in containers, or managed by resource frameworks like YARN or Mesos. TaskManagers connect to JobManagers, announcing themselves as available, and are assigned work. Figure 4 demonstrates the main components of Flink framework.

2.4. Storm. Storm [17] is a free open-source distributed stream processing computation framework. It takes several characteristics from the popular actor model and can be used with practically any kind of programming language for developing applications such as real-time streaming analytics, critical work flow systems, and data delivery services. The engine may process billions of tuples each day in a fault-tolerant way. It can be integrated with popular resource management frameworks such as YARN, Mesos, and Docker. Apache Storm cluster is made up of two types of processing actors: spouts and bolts.

(i) Spout is connected to the external data source of a stream and is continuously emitting or collecting new data for further processing.

(ii) Bolt is a processing logic unit within a streaming processing topology; each bolt is responsible for a
certain processing task such as transformation, filtering, aggregating, and partitioning.

Storm defines workflow as directed acyclic graphs (DAGs), called topologies with connected spouts and bolts as vertices. Edges in the graph define the link between the bolts and the data stream. Unlike batch jobs being only executed once, Storm jobs run forever until they are killed. There are two types of nodes in a Storm cluster: nimbus (master node) and supervisor (worker node). Nimbus, similar to Hadoop JobTracker, is the core component of Apache Storm and is responsible for distributing load across the cluster, queuing and assigning tasks to different processing units, and monitoring execution status. Each worker node executes a process known as the supervisor which may have one or more worker processes. Supervisor delegates the tasks to worker processes. Worker process then creates a subset of topology to run the task. Apache Storm does rely on an internal distributed messaging system, called Netty, for the communication between nimbus and supervisors. Zookeeper manages the communication between real-time job trackers (nimbus) and supervisors (Storm workers). Figure 5 outlines the high-level view of Storm cluster.

2.5. Apache Samza. Apache Samza [18] is a distributed stream processing framework, mainly written in Scala and Java. Overall, it has a relatively high throughput as well as somewhat increased latency when compared to Storm [8]. It uses Apache Kafka, which was originally developed for LinkedIn, for messaging and streaming, while Apache Hadoop YARN/Mesos is utilized as an execution platform for overall resource management. Samza relies on Kafka’s semantics to define the way streams are handled. Its main objective is to collect and deliver massively large volumes of event data, in particular, log data with a low latency. A Kafka system’s architecture is comparatively simple as it only consists of a set of brokers which are individual nodes that make up a Kafka cluster. Data streams are defined by topics, which is a stream of related information that consumers can subscribe to. Topics are divided into partitions that are distributed over the broker instances for retrieving the corresponding messages using a pull mechanism. The basic flow of job execution is presented in Figure 6.

Tables 1 and 2 present a brief comparative analysis of these frameworks based on some common attributes. As shown in the tables, MapReduce computation data flow follows chain of stages with no loop. At each stage, the program
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Hadoop</th>
<th>Spark</th>
<th>Storm</th>
<th>Samza</th>
<th>Flink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current stable version</td>
<td>2.8.1</td>
<td>2.2.0</td>
<td>1.1.1</td>
<td>0.13.0</td>
<td>1.3.2</td>
</tr>
<tr>
<td>Batch processing</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Computational model</td>
<td>MapReduce</td>
<td>Streaming (microbatches)</td>
<td>Streaming (microbatches)</td>
<td>Streaming</td>
<td>Supports continuous flow streaming, microbatch, and batch</td>
</tr>
<tr>
<td>Data flow</td>
<td>Chain of stages</td>
<td>Directed acyclic graph</td>
<td>Directed acyclic graphs (DAGs) with spouts and bolts</td>
<td>Streams (acyclic graph)</td>
<td>Controlled cyclic dependency graph through machine learning</td>
</tr>
<tr>
<td>Resource management</td>
<td>YARN</td>
<td>YARN/Mesos</td>
<td>HDFS (YARN)/Mesos</td>
<td>YARN/Mesos</td>
<td>Zookeeper/YARN/Mesos</td>
</tr>
<tr>
<td>Language support</td>
<td>All major languages</td>
<td>Java, Scala, Python, and R</td>
<td>Any programming language</td>
<td>JVM languages</td>
<td>Java, Scala, Python, and R</td>
</tr>
<tr>
<td>Job management/optimization</td>
<td>MapReduce approach</td>
<td>Catalyst extension</td>
<td>Storm-YARN/3rd-party tools like Ganglia</td>
<td>Internal JobRunner</td>
<td>Internal optimizer</td>
</tr>
<tr>
<td>Interactive mode</td>
<td>None (3rd-party tools like Impala can be integrated)</td>
<td>Interactive shell</td>
<td>None</td>
<td>Limited API of Kafka streams</td>
<td>Scala shell</td>
</tr>
<tr>
<td>Machine learning libraries</td>
<td>Apache Mahout/H2O</td>
<td>Spark ML and MLib</td>
<td>Trident-ML/Apache SAMOA</td>
<td>Apache SAMOA</td>
<td>Flink-ML</td>
</tr>
<tr>
<td>Maximum reported nodes (scalability)</td>
<td>Yahoo Hadoop cluster with 42,000 nodes</td>
<td>8000</td>
<td>300</td>
<td>LinkedIn with around a hundred node clusters</td>
<td>Alibaba customized Flink cluster with thousands of nodes</td>
</tr>
</tbody>
</table>
Table 2: Comparative analysis of big data resource frameworks (s = 5).

<table>
<thead>
<tr>
<th>Feature</th>
<th>Hadoop</th>
<th>Spark</th>
<th>Flink</th>
<th>Storm</th>
<th>Samza</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing speed</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>Scalability</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>Machine learning</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>Low latency</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>Security</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>Dataset size</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
</tbody>
</table>

Figure 5: Architecture of Storm Cluster, source: [17].

Figure 6: Samza architecture, source: [18].

proceeds with the output from the previous stage and generates an input for the next stage. Although machine learning algorithms are mostly designed in the form of cyclic data flow, Spark, Storm, and Samza represent it as directed acyclic graph to optimize the execution plan. Flink supports controlled cyclic dependency graph in runtime to represent the machine learning algorithms in a very efficient way. Hadoop and Storm do not provide any default interactive environment. Apache Spark has a command-line interactive shell to use the application features. Flink provides a Scala shell to configure standalone as well as cluster setup. Apache Hadoop is highly scalable and it has been used in the Yahoo production consisting of 42000 nodes in 20 YARN clusters. The largest known cluster size for Spark is of 8000 computing nodes while Storm has been tested on a maximum of 300 node clusters. Apache Samza cluster, with around a hundred nodes, has been used in LinkedIn data flow and application messaging system. Apache Flink has been customized for Alibaba search engine with a deployment capacity of thousands of processing nodes.

3. Comparative Evaluation of Big Data Frameworks

Big data in cloud computing, a popular research trend, is posing significant influence on current enterprises, IT industries, and research communities. There are a number of disruptive and transformative big data technologies and solutions that are rapidly emanating and evolving in order to provide data-driven insight and innovation. Furthermore, modern cloud computing services are offering all kinds of big data analytic tools, technologies, and computing infrastructures to speed up the data analysis process at an affordable cost. Although many distributed resource management frameworks are available nowadays, the main issue is how to
select a suitable big data framework. The selection of one big
data platform over the others will come down to the specific
application requirements and constraints that may involve
several tradeoffs and application usage scenarios. However,
we can identify some key factors that need to be fulfilled
before deploying a big data application in the cloud. In this
section, based on some empirical evidence from the available
literature, we discuss the advantages and disadvantages of
each resource management framework.

3.1. Processing Speed. Processing speed is an important per-
formance measurement that may be used to evaluate the
effectiveness of different resource management frameworks.
It is a common metric for the maximum number of I/O oper-
ations to disk or memory or the data transfer rate between
the computational units of the cluster over a specific amount of
time. Based on the context of big data, the average processing
speed represented as \( \overline{m} \), calculated after \( n \) iterations run, is the
maximum amount of memory/disk intensive operations that
can be performed over a time interval \( t_i \):

\[
\overline{m} = \frac{\sum_{i=1}^{n} m_i}{\sum_{i=1}^{n} t_i} \quad (1)
\]

Veiga et al. [30] conducted a series of experiments on a
multicore cluster setup to demonstrate performance results of
Apache Hadoop, Spark, and Flink. Apache Spark and
Flink resulted to be much efficient execution platforms over
Hadoop while performing nonsort benchmarks. It was fur-
ther noted that Spark showed better performance results for
operations such as WordCount and K-Means (CPU-bound in
nature) while Flink achieved better results in PageRank algo-
rithm (memory bound in nature). Mavridis and Karatza [31]
experimentally compared performance statistics of Apache
Hadoop and Spark on Okeanos IaaS cloud platform. For each
set of experiments, necessary statistics related to execution
time, working nodes, and the dataset size were recorded.
Spark performance was found optimal as compared to
Hadoop for most of the cases. Furthermore, Spark on YARN
platform showed suboptimal results as compared to the case
when it was executed in standalone mode. Some similar
results were also observed by Zaharia et al. [32] on a 100 GB
dataset record. Vellaapandiyan and Raja [33] demonstrated
performance evaluation and comparison of Hadoop and
Spark frameworks on resident’s record dataset ranging from
100 GB to 900 GB of size. Spark scale of performance was
relatively better when the dataset size was between small and
medium size (100 GB–750 GB); afterwards, its performance
decayed as compared to Hadoop. The primary reason for the
performance decline was evident as Spark cache size could
not fit into the memory for the larger dataset. Taran et al.
[34] quantified performance differences of Hadoop and Spark
using WordCount dataset which was ranging from 100 KB
to 1 GB. It was observed that Hadoop framework was five
times faster than Spark when the evaluation was performed
using a larger set of data sources. However, for the smaller
tasks, Spark showed better performance results. However, the
speed-up ratio was decreased for both databases with the
growth of input dataset.

Gopalani and Arora [35] used K-Means algorithm on
some small, medium, and large location datasets to compare
Hadoop and Spark frameworks. The study results showed that
Spark performed up to three times better than MapReduce
for most of the cases. Bertoni et al. [36] performed the
experimental evaluation of Apache Flink and Storm using
large genomic dataset data on Amazon EC2 cloud. Apache
Flink was superior to Storm while performing histogram
and map operations while Storm outperformed Flink while
 genomic join application was deployed.

3.2. Fault Tolerance. Fault tolerance is the characteristic
that enables a system to continue functioning in case of
the failure of one or more components. High-performance
computing applications involve hundreds of nodes that are
interconnected to perform a specific task; failing a node
should have zero or minimal effect on overall computation.
The tolerance of a system, represented as \( \text{Tol}_{FT} \), to meet its
requirements after a disruption is the ratio of the time to
complete tasks without observing any fault events to the
overall execution time where some fault events were detected
and the system state is reverted back to consistent state:

\[
\text{Tol}_{FT} = \frac{T_x}{\overline{T_x} + \sigma^2}, \quad (2)
\]

where \( T_x \) is the estimated correct execution time obtained
from a program run that is presumed to be fault-free, or by
averaging the execution time from several application runs
that produce a known correct output, and \( \sigma^2 \) represents the
variance in a program's execution time due to the occurrence
of fault events. For an HPC application that consists of a set
of computationally intensive tasks \( \Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\} \) and since
\( \text{Tol}_{FT} \) for each individual task is computed as \( \text{Tol}_{\tau_i} \), then the
overall application resilience, \( \text{Tol} \), may be calculated as [37]

\[
\text{Tol} = \sqrt{\text{Tol}_{\tau_1} \cdot \text{Tol}_{\tau_2} \cdot \ldots \cdot \text{Tol}_{\tau_n}}, \quad (3)
\]

Lu et al. [38] used StreamBench toolkit to evaluate perform-
ance and fault tolerance ability of Apache Spark, Storm,
Spark, and Samza. It was found that, with the increased data
size, Spark is much stable and fault-tolerant as compared to
Storm but may be less efficient when compared to Samza.
Furthermore, when compared in terms of handling large
capacity of data, both Samza and Spark outperformed Storm.
Gu and Li [39] used PageRank algorithm to perform a
comparative experiment on Hadoop and Spark frameworks.
It was observed that, for smaller datasets such as wiki-Vote
and soc-Slashdot0902, Spark outperformed Hadoop with a
much better margin. However, this speed-up result degraded
with the growth of dataset, and for large datasets, Hadoop
easily outperformed Spark. Furthermore, for massively large
datasets, Spark was reported to be crashed with JVM heap
exception while Hadoop still performed its task. Lopez et al.
[40] evaluated throughput and fault tolerance mechanism of
Apache Storm and Flink. The experiments were based on a
threat detection system where Apache Storm demonstrated
better throughput as compared to Flink. For fault tolerance,
different virtual machines were manually turned off to analyze the impact of node failures. Apache Flink used its internal subsystem to detect and migrate the failed tasks to other machines and hence resulted in very few message losses. On the other hand, Storm took more time as Zookeeper, involving some performance overhead, was responsible for reporting the state of nimbus and thereafter processing the failed task on other nodes.

3.3. Scalability. Scalability refers to the ability to accommodate large loads or change in size/workload by provisioning of resources at runtime. This can be further categorized as scale-up (by making hardware stronger) or scale-down (by adding additional nodes). One of the critical requirements of enterprises is to process large volumes of data in a timely manner to address high-value business problems. Dynamic resource scalability allows business entities to perform massive computation in parallel, thus reducing overall time, complexity, and effort. The definition of scalability comes from Amdahl’s and Gustafson’s laws [41]. Let \( W \) be the size of workload before the improvement of the system resources; the fraction of the execution workload that benefits from the improvement of system resources is \( \alpha \) and the fraction concerning the part that would not benefit from improvement in the resources is \( 1 - \alpha \). When using an \( n \)-processor system, user workload is scaled to

\[
\hat{W} = \alpha W + (1-\alpha)nW. \tag{4}
\]

The parallel execution time of a scaled workload \( \hat{W} \) on \( n \)-processors is defined as scaled-workload speed-up \( \hat{S} \) as shown in

\[
\hat{S} = \frac{\hat{W}}{W} = \frac{\alpha W + (1-\alpha)nW}{W}. \tag{5}
\]

García-Gil et al. [42] performed scalability and performance comparison of Apache Spark and Flink using feature selection framework to assemble multiple information theoretic criteria into a single greedy algorithm. The ECBDL14 dataset was used to measure scalability factor for the frameworks. It was observed that Spark scalability performance was 4–10 times faster than Flink. Jakovits and Srirama [43] analyzed four MapReduce based frameworks including Hadoop and Spark for benchmarking partitioning around Medoids, Clustering Large Applications, and Conjugate Gradient linear system solver algorithms using MPI. All experiments were performed on 33 Amazon EC2 large instances cloud. For all algorithms, Spark performed much better as compared to Hadoop, in terms of both performance and scalability. Boden et al. [44] aimed to investigate the scalability with respect to both data size and dimensionality in order to demonstrate a fair and insightful benchmark that reflects the requirements of real-world machine learning applications. The benchmark was comprised of distributed optimization algorithms for supervised learning as well as algorithms for unsupervised learning. For supervised learning, they implemented machine learning algorithms by using Breeze library, while for unsupervised learning they chose the popular \( k \)-Means clustering algorithm in order to assess the scalability of the two resource management frameworks. The overall execution time of Flink was relatively low on the resource-constrained settings with a limited number of nodes, while Spark had a clear edge once enough main memory was available due to the addition of new computing nodes.

3.4. Machine Learning and Iterative Tasks Support. Big data applications are inherently complex in nature and usually involve tasks and algorithms that are iterative in nature. These applications have distinct cyclic nature to achieve the desired result by continually repeating a set of tasks until these cannot be substantially reduced further.

Spangenberg et al. [45] used real-world datasets, consisting of four algorithms, that is, WordCount, \( K \)-Means, PageRank, and relational query, to benchmark Apache Flink and Storm. It was observed that Apache Storm performs better in batch mode as compared to Flink. However, with the increasing complexity, Apache Flink had a performance advantage over Storm and thus it was better suited for iterative data or graph processing. Shi et al. [46] focused on analyzing Apache MapReduce and Spark for batch and iterative jobs. For smaller datasets, Spark resulted to be a better choice, but when experiments were performed on larger datasets, MapReduce turned out to be several times faster than Spark. For iterative operations such as \( K \)-Means, Spark turned out to be 1.5 times faster as compared to MapReduce in its first iteration, while Spark was more than 5 times faster in subsequent operations.

Kang and Lee [47] examined five resource management frameworks including Apache Hadoop and Spark with respect to performance overheads (disk input/output, network communication, scheduling, etc.) in supporting iterative computation. The PageRank algorithm was used to evaluate these performance issues. Since static data processing tends to be a more frequent operation than dynamic data as it is used in every iteration of MapReduce, it may cause significant performance overhead in case of MapReduce. On the other hand, Apache Spark uses read-only cached version of objects (resilient distributed dataset) which can be reused in parallel operations, thus reducing the performance overhead during iterative computation. Lee et al. [48] evaluated five systems including Hadoop and Spark over various workloads to compare against four iterative algorithms. The experimentation was performed on Amazon EC2 cloud. Overall, Spark showed the best performance when iterative operations were performed in main memory. In contrast, the performance of Hadoop was significantly poor as compared to other resource management systems.

3.5. Latency. Big data and low latency are strongly linked. Big data applications provide true value to businesses, but these are mostly time critical. If cloud computing has to be the successful platform for big data implementation, one of the key requirements will be the provisioning of high-speed network to reduce communication latency. Furthermore, big data frameworks usually involve centralized design where the scheduler assigns all tasks through a single node which may significantly impact the latency when the size of data is huge.
Let $T_{\text{elapsed}}$ be the elapsed time between the start and finish time of a program in a distributed architecture, $T_j$ be the effective execution time, and $\lambda_j$ be the sum of total idle units of $i$th processor from a set of $N$ processors. Then, the average latency, represented as $\lambda(W, N)$, for the size of workload $W$, is defined as the average amount of overhead time needed for each processor to complete the task:

$$
\lambda(W, N) = \frac{\sum_{i=1}^{N} (T_{\text{elapsed}} - T_j + \lambda_j)}{N}.
$$

(6)

Chintapalli et al. [49] conducted a detailed analysis of Apache Storm, Flink, and Spark streaming engines for latency and throughput. The study results indicated that, for high throughput, Flink and Storm have significantly lower latency as compared to Spark. However, Spark was able to handle high throughput as compared to other streaming engines. Lu et al. [38] proposed StreamBench benchmark framework to evaluate modern distributed stream processing frameworks. The framework includes dataset selection, data generation methodologies, program set description, workload suites design, and metric proposition. Two real-world datasets, AOL Search Data and CAIDA Anonymized Internet Traces Dataset, were used to assess performance, scalability, and fault tolerance aspects of streaming frameworks. It was observed that Storm's latency in most cases was far less than Spark's except in the case when the scales of the workload and dataset were massive in nature.

3.6. Security. Instead of the classical HPC environment, where information is stored in-house, many big data applications are now increasingly deployed on the cloud where privacy-sensitive information may be accessed or recorded by different data users with ease. Although data privacy and security issues are not a new topic in the area of distributed computing, their importance is amplified by the widespread adoption of cloud computing services for big data platforms. The dataset may be exposed to multiple users for different purposes which may lead to security and privacy risks.

Let $N$ be a list of security categories that may be provided for security mechanism. For instance, a framework may use encryption mechanism to provide data security and access control list for authentication and authorization services. Let $\max(W_j)$ be the maximum weight that is assigned to the $i$th security category from a list of $N$ categories and $W_j$ be the reputation score of a particular resource management framework. Then, the framework security ranking score can be represented as

$$
\text{Secscore} = \frac{\sum_{i=1}^{N} W_j}{\sum_{i=1}^{N} \max(W_j)}.
$$

(7)

Hadoop and Storm use Kerberos authentication protocol for computing nodes to provide their identity [50]. Spark adopts a password-based shared secret configuration as well as Access Control Lists (ACLs) to control the authentication and authorization mechanisms. In Flink, stream brokers are responsible for providing authentication mechanism across multiple services. Apache Samza/Kafka provides no built-in security at the system level.

3.7. Dataset Size Support. Many scientific applications scale up to hundreds of nodes to process massive amounts of data that may exceed over hundreds of terabytes. Unless big data applications are properly optimized for larger datasets, this may result in performance degradation with the growth of the data. Furthermore, in many cases, this may result in a crash of software, resulting in loss of time and money. We used the same methodology to collect big data support statistics as presented in earlier sections.

4. Discussion on Big Data Framework

Every big data framework has been evolved for its unique design characteristics and application-specific requirements. Based on the key factors and their empirical evidence to evaluate big data frameworks, as discussed in Section 3, we produce the summary of results of seven evaluation factors in the form of star ranking, Ranking$_{RF} \in [0, s]$, where $s$ represents the maximum evaluation score. The general equation to produce the ranking, for each resource framework (RF), is given as

$$
\text{Ranking}_{RF} = \frac{\sum_{i=1}^{7} \sum_{j=1}^{N} \max(W_{ij}) \times \sum_{i=1}^{7} \sum_{j=1}^{N} W_{ij}}{s},
$$

(8)

where $N$ is the total number of research studies for a particular set of evaluation metrics, $\max(W_{ij}) \in [0, 1]$ is the relative normalized weight assigned to each literature study based on the number of experiments performed, and $W_{ij}$ is the framework test-bed score calculated from the experimentation results from each study.

Hadoop MapReduce has a clear edge on large-scale deployment and larger dataset processing. Hadoop is highly compatible and interoperable with other frameworks. It also offers a reliable fault tolerance mechanism to provide a failure-free mechanism over a long period of time. Hadoop can operate on a low-cost configuration. However, Hadoop is not suitable for real-time applications. It has a significant disadvantage when latency, throughput, and iterative job support for machine learning are the key considerations of application requirements.

Apache Spark is designed to be a replacement for batch-oriented Hadoop ecosystem to run-over static and real-time datasets. It is highly suitable for high-throughput streaming applications where latency is not a major issue. Spark is memory intensive and all operations take place in memory. As a result, it may crash if enough memory is not available for further operations (before the release of Spark version 1.5, it was not capable of handling datasets larger than the size of RAM and the problem of handling larger dataset still persists in the newer releases with different performance overheads). Few research efforts, such as Project Tungsten, are aimed at addressing the efficiency of memory and CPU for Spark applications. Spark also lacks its own storage system so its integration with HDFS through YARN or Cassandra using Mesos is an extra overhead for cluster configuration.

Apache Flink is a true streaming engine. Flink supports both batch and real-time operations over a common runtime to fulfill the requirements of Lambda architecture. However,
it may also work in batch mode by stopping the streaming source. Like Spark, Flink performs all operations in memory, but in case of memory hog, it may also use disk storage to avoid application failure. Flink has some major advantages over Hadoop and Spark by providing better support for iterative processing with high throughput at the cost of low latency.

Apache Storm was designed to provide a scalable, fault tolerance, real-time streaming engine for data analysis, which Hadoop did for batch processing. However, the empirical evidence suggests that Apache Storm proved to be inefficient to meet the scale-up/scale-down requirements for real-time big data applications. Furthermore, since it uses microbatch stream processing, it is not very efficient where continuous stream process is a major concern, nor does it provide a mechanism for simple batch processing. For fault tolerance, Storm uses Zookeeper to store the state of the processes which may involve some extra overhead and may also result in message loss. On the other hand, Storm is an ideal solution for near-real-time application processing where workload could be processed with a minimal delay with strict latency requirements.

Apache Samza, in integration Kafka, provides some unique features that are not offered by other stream processing engines. Samza provides a powerful check-pointing based fault tolerance mechanism with minimal data loss. Samza jobs can have high throughput with low latency when integrated with Kafka. However, Samza lacks some important features as data processing engine. Furthermore, it offers no built-in security mechanism for data access control.

To categorize the selection of best resource engine based on a particular set of requirements, we use the framework proposed by Chung et al. [51]. The framework provides a layout for matching, ranking, and selecting a system based on some particular requirements. The matching criterion is based on user goals which are categorized as soft and hard goals. Soft goals represent the nonfunctional requirements of the system (such as security, fault tolerance, and scalability) while hard goals represent the functional aspects of the system (such as machine learning and data size support). The relationship between the system components and the goals can be ranked as very positive (++) positive (+), negative (−), and very negative (−−). Based on the evidence provided from the literature, the categorization of major resource management frameworks is presented in Figure 7.

5. Related Work

Our research work differs from other efforts because the subject goal and object of study are not identical, as we provide an in-depth comparison of popular resource engines based on empirical evidence from existing literature. Hesse and Lorenz [8] conducted a conceptual survey on stream processing systems. However, their discussion was focused on some basic differences related to real-time data processing engines. Singh and Reddy [9] provided a thorough analysis of big data analytic platforms that included peer-to-peer networks, field programmable gate arrays (FPGA), Apache Hadoop ecosystem, high-performance computing (HPC) clusters, multicore CPU, and graphics processing unit (GPU). Our case is different here as we are particularly interested in big data processing engines. Finally, Landset et al. [10] focused on machine learning libraries and their evaluation based on ease of use, scalability, and extensibility. However, our work differs from their work as the primary focuses of both studies are not identical.

Chen and Zhang [11] discussed big data problems, challenges, and associated techniques and technologies to address these issues. Several potential techniques including cloud computing, quantum computing, granular computing, and biological computing were investigated and the possible opportunities to explore these domains were demonstrated. However, the performance evaluation was discussed only on theoretical grounds. A taxonomy and detailed analysis of the state of the art in big data 2.0 processing systems were presented in [12]. The focus of the study was to identify current research challenges and highlight opportunities for new innovations and optimization for future research and development. Assunção et al. [13] reviewed multiple generations of data stream processing frameworks that provide mechanisms for resource elasticity to match the demands of stream processing services. The study examined the challenges associated with efficient resource management decisions and suggested solutions derived from the existing research studies. However, the study metrics are restricted to the elasticity/scalability aspect of big data streaming frameworks.

As shown in Table 3, our work differs from the previous studies which focused on the classification of resource management frameworks on theoretical grounds. In contrast to the earlier approaches, we classify and categorize big data resource management frameworks based on empirical grounds, derived from multiple evaluation/experimentation studies. Furthermore, our evaluation/ranking methodology is based on a comprehensive list of study variables which were not addressed in the studies conducted earlier.

6. Conclusions and Future Work

There are a number of disruptive and transformative big data technologies and solutions that are rapidly emanating and evolving in order to provide data-driven insight and innovation. The primary object of the study was how to classify popular big data resource management systems. This study was also aimed at addressing the selection of candidate resource provider based on specific big data application requirements. We surveyed different big data resource management frameworks and investigated the advantages and disadvantages for each of them. We carried out the performance evaluation of resource management engines based on seven key factors and each one of the frameworks was ranked based on the empirical evidence from the literature.

6.1. Observations and Findings. Some key findings of the study are as follows:

(i) In terms of processing speed, Apache Flink outperforms other resource management frameworks for small, medium, and large datasets [30, 36]. However, during our own set of experiments on Amazon EC2
<table>
<thead>
<tr>
<th>Study reference</th>
<th>Data model</th>
<th>Resource frameworks</th>
<th>Study features</th>
<th>Evaluation/ranking methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>[8]</td>
<td>Data stream processing systems</td>
<td>Storm, Flink, Spark, Samza</td>
<td>A brief comparison of resource frameworks</td>
<td>×</td>
</tr>
<tr>
<td>[9]</td>
<td>Batch and stream processing systems</td>
<td>Horizontal scaling systems, such as peer-to-peer, MapReduce/MPI, and Spark, and vertical scaling systems, such as CUDA and HDL</td>
<td>Comparison of horizontal and vertical scaling systems</td>
<td>Theoretical comparison of resource frameworks</td>
</tr>
<tr>
<td>[10]</td>
<td>Batch and stream processing engines</td>
<td>MapReduce, Spark, Flink, and Storm as well as machine learning libraries</td>
<td>Machine learning libraries and their evaluation mechanism</td>
<td>Performance comparison with respect to machine learning toolkits</td>
</tr>
<tr>
<td>[11]</td>
<td>Batch and stream processing frameworks</td>
<td>Hadoop, Storm, and other big data frameworks</td>
<td>In-depth analysis of big data opportunities and challenges</td>
<td>×</td>
</tr>
<tr>
<td>[12]</td>
<td>Batch and stream processing frameworks</td>
<td>Hadoop, Spark, Storm, Flink, and Tez as well as SQL, Graph, and bulk synchronous parallel model</td>
<td>Analysis of current open research challenges in the field of big data and the promising directions for future research</td>
<td>×</td>
</tr>
<tr>
<td>[13]</td>
<td>Stream processing engines</td>
<td>Apache Storm, S4, Flink, Samza, Spark Streaming, and Twitter Heron</td>
<td>Classification of elasticity metrics for resource allocation strategies that meet the demands of stream processing services</td>
<td>Evaluation of elasticity/scaling metrics for stream processing systems</td>
</tr>
</tbody>
</table>
cluster with varied task managers settings (1–4 task managers per node), Flink failed to complete custom smaller size JVM dataset jobs due to inefficient memory management of Flink memory manager. We could not find any reported evidence of this particular use case in these relevant literature studies. It seems that most of the performance evaluation studies employed standard benchmarking test sets where dataset size was relatively large and hence this particular use case was not reported in these studies. Further research effort is required to elucidate the underlying specific factors under this particular case.

(ii) Big data applications usually involve massive amount of data. Apache Spark supports necessary strategies for fault tolerance mechanism, but it has been reported to crash on larger datasets. Even during our experimentations, Apache Spark version 1.6 (selected due to the compatibility reasons with earlier researches) crashed on several occasions when the dataset was larger than 500 GB. Although Spark has been ranked higher in terms of fault tolerance with the increase of data scale in several studies [38, 52], it has limitations in handling larger typical big data dataset applications and hence such studies cannot be generalized.

(iii) Spark MLlib and Flink-ML offer a variety of machine learning algorithms and utilities to exploit distributed and scalable big data applications. Spark MLlib outperforms Flink-ML in most of the machine learning use cases [42], except in the case when repeated passes are performed on unchanged data. However, this performance evaluation may further be investigated.

Figure 7: Categorization of resource engines based on key big data application requirements.
as research studies such as [53], reported differently where Flink outperformed Spark on sufficiently large cases.

(iv) For graph processing algorithms such as PageRank, Flink uses Gelly library that provides native closed-loop iteration operators, making it a suitable platform for large-scale graph analytics. Spark, on the other hand, uses GraphX library that has much longer pre-processing time to build graph and other data structures, making its performance worse as compared to Apache Flink. Apache Flink has been reported to obtain the best results for graph processing datasets (3x–5x in [54] and 2x-3x in [45]) as compared to Spark. However, some studies such as [55] reported Spark to be 1.7x faster than Flink for large graph processing. Such inconsistent behavior may be further investigated in the future research studies.

6.2. Future Work. In the area of big data, there is still a clear gap that requires more effort from the research community to build in-depth understanding of performance characteristics of big data resource management frameworks. We consider this study a step towards enlarging our knowledge to understand the big data world and provide an effort towards the direction of improving the state of the art and achieving the big vision on the big data domain. In earlier studies, a clear ranking cannot be established as the study parameters were mostly limited to a few issues such as throughput, latency, and machine learning. Furthermore, further investigation is required on resource engines such as Apache Samza in comparison with other frameworks. In addition, research effort needs to be carried out in several areas such as data organization, platform specific tools, and technological issues in big data domain in order to create next-generation big data infrastructures. The performance evaluation factors might also vary among systems depending on the used algorithms. As future work, we plan to benchmark these popular resource engines for meeting resource demands and requirements for different scientific applications. Moreover, a scalability analysis could be done. Particularly, the performance evaluation of adding dynamic worker nodes and the resulting performance analysis is of peculiar interest. Additionally, further research can be carried out in order to evaluate performance aspects with respect to resource competition between jobs (on different research schedules such as YARN and Mesos) and the fluctuation of available computing resources. Finally, most of the experiments in earlier studies were performed using standard parameter configurations; however, each resource management framework offers domain specific tweaks and configuration optimization mechanisms for meeting application-specific requirements. The development of a benchmark suite that aims to find maximum throughput based on configuration optimization would be an interesting direction of future research.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[14] "Big data working group big data taxonomy, 2014”.


N. Yuhanna and M. Gualtieri, "Elasticity, Automation, and Pay-As-You-Go Compel Enterprise Adoption Of Hadoop in the Cloud, Q2 2016".


in Proceedingsof the 25th ACM International Conference on
Information and Knowledge Management, CIKM 2016, pp.
2089–2094, October 2016.
[49] S. Chintapalli, D. Dagit, B. Evans et al., "Benchmarking stream-
ing computation engines: Storm, Flink and Spark streaming," in Proceedingsof the 30th IEEE International Parallel and
Distributed Processing Symposium Workshops, IPDPSW 2016,

preserving layer over MapReduce on cloud," in Proceedingsof the 2nd International Conference on Cloud and Green Comput-
ing, CGC 2012, Held Jointly with the 2nd International Con-
304–310, chin, November 2012.

[51] L. Chung, B. A. Nixon, and E. Yu, "Using nonfunctional require-
ments to systematically select among alternatives in architec-
tural design," in Proceedingsof the 1st International Workshop

[52] S. Qian, G. Wu, J. Huang, and T. Das, "Benchmarking modern
distributed streaming platforms," in Proceedingsof the 2016 IEEE
International Conference on Industrial Technology (ICIT),

[53] F. Dambrreville and A. Toumi, "Generic and massively concurren-
t computation of belief combination rules," in Proceedingsof the the International Conference on Big Data and Advanced
Wireless Technologies, pp. 1–6, Blagoevgrad, Bulgaria, November
2016.

[54] R. W. Techentin, M. W. Markland, R. J. Poole, D. R. H. C. R.
Haider, and B. K. Gilbert, "Page rank performance evaluation
of cluster computing frameworks on Cray Urika-GX supercom-
puter," Computer Science and Engineering, vol. 6, pp. 33–38,
2016.

[55] O.-C. Marcus, A. Costan, G. Antoniu, and M. S. Pérez-
Hernández, "Spark versus flink: understanding performance in
big data analytics frameworks," in Proceedingsof the 2016 IEEE
International Conference on Cluster Computing. CLUSTER 2016,
NUMA-Aware Thread Scheduling for Big Data Transfers over Terabits Network Infrastructure

Taeuk Kim, Awais Khan, Youngjae Kim, Preethika Kasu, and Scott Atchley

1 Sogang University, Seoul, Republic of Korea
2 Ajou University, Suwon, Republic of Korea
3 Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

Correspondence should be addressed to Youngjae Kim; youkim@sogang.ac.kr

Received 29 December 2017; Accepted 27 March 2018; Published 7 May 2018

Copyright © 2018 Taeuk Kim et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The evergrowing trend of big data has led scientists to share and transfer the simulation and analytical data across the geodistributed research and computing facilities. However, the existing data transfer frameworks used for data sharing lack the capability to adopt the attributes of the underlying parallel file systems (PFS). LADS (Layout-Aware Data Scheduling) is an end-to-end data transfer tool optimized for terabit network using a layout-aware data scheduling via PFS. However, it does not consider the NUMA (Nonuniform Memory Access) architecture. In this paper, we propose a NUMA-aware thread and resource scheduling for optimized data transfer in terabit network. First, we propose distributed RMA buffers to reduce memory controller contention in CPU sockets and then schedule the threads based on CPU socket and NUMA nodes inside CPU socket to reduce memory access latency. We design and implement the proposed resource and thread scheduling in the existing LADS framework. Experimental results showed from 21.7% to 44% improvement with memory-level optimizations in the LADS framework as compared to the baseline without any optimization.

1. Introduction

The continuous inflation in data generation is raising the sharing and collaboration needs for effective simulations and real-time analysis. Such sharing and collaboration need a massive-scale data transfer across geodispersed data centers [1]. The Brookhaven National Lab (BNL) cooperates with European Large Hadron Collider (LHC) in the ATLAS experiment where more than 3,000 scientists participate and produce petabytes of simulation and analytical data, motivating collaboration ventures [2]. Such large-scale collaboration environments highly motivate us to revisit the architecture of existing end-to-end data transfer tools such as bbcp [3], LADS [1], and GridFTP [4].

In an end-to-end data transfer between geodispersed data centers, there are three significant factors concerning the data transfer performance and throughput, that is, network, storage, and memory. However, in realistic large-scale HPC environments, the network infrastructure delivers a high bandwidth and it is further improving [5], for example, ESnet of DOE [6]. So, we do not consider network as a major bottleneck in limiting the data transfer activities in large-scale HPC environments equipped with high-speed network connectivity [5, 6].

The parallel file system (PFS) [7, 8] based storage-backends deployed in the data centers act as a bottleneck when thread count exceeds the service rate of object storage server (OSS) or multiple threads access the same object storage target (OST). LADS [1], high-speed end-to-end data transfer tool between data centers, minimizes this I/O contention by being aware of data chunk's layout and scheduling threads based on it. The memory bottleneck can be incurred in NUMA environment when threads access remote NUMA node's memory. If the buffer which LADS uses to transfer data is allocated in a different NUMA node from I/O threads, I/O threads need to access remote NUMA node during data transfer and it makes memory access consume longer latency. Currently, LADS tool does not offer any solution to overcome...
this problem. In this paper, we propose MTS (Memory-aware Thread Scheduling) method to solve the memory bottleneck issues with LADS data transfer tool.

The data transfer frameworks such as GridFTP [4], bbcp [3], and LADS [1] have been designed to ensure high-speed data transmission. However, GridFTP and bbcp are designed on file-based data transfer, whereas LADS is optimized for object-based data transfer where multiple threads can work on multiple object chunks simultaneously to improve end-to-end data transfer speed. With the improvement in network infrastructure, the parallel and distributed file systems such as Lustre [8], Gluster [7], and Ceph [9] are improving their storage and computing frameworks in order to derive the maximum bandwidth. GridFTP [4] and bbcp [3] cannot gain high benefits from these file systems since they are designed without considering the underlying file system, whereas LADS [1], due to its layout-aware nature of the data transfer mechanism, can fully utilize the benefits of these underlying parallel file systems. Besides, LADS uses Common Communication Interface (CCI) [5] to exploit the high-speed terabit networks.

In this paper, we emphasize the possible bottlenecks and opportunities such as high-speed network and NUMA architecture in end-to-end data transfer path. An end-to-end data transfer can meet multiple bottlenecks such as (i) storage, (ii) CPU, and (iii) memory. The storage becomes a bottleneck when data transfer software is unaware of the underlying file system architecture; for example, parallel file systems use chunking and striping techniques to store data in a more efficient fashion. So, the data transfer tool can efficiently utilize storage bandwidth by having the knowledge about storage layout, without which storage bandwidth can be underutilized. The CPU bottleneck occurs, if data transfer tool does not take into account the utilization of multicores while transferring data. The underutilization can happen when the threads are less than the CPU cores and some of the cores remain idle. The overutilization can occur, where thread count is very high with respect to available CPU cores.

The memory contention can occur in two conditions. Firstly, when multiple threads or processes are accessing the same shared memory region. Secondly, when threads hosted on CPU sockets access the remote memory of other CPU sockets. Both these contentions can lead to increased memory access latency. LADS framework addresses the storage and CPU bottlenecks by implementing layout-aware multithreaded architecture [1]. However, LADS does not consider the memory bottleneck issues.

In this paper, we address the memory bottleneck issues by proposing memory buffer partitioning at each CPU socket and scheduling threads with being aware of NUMA architecture. Partitioning the memory buffer reduces the shared memory regions and thread scheduling reduces the remote memory access across CPU sockets.

This paper makes the following contributions:

(i) Increase in the number of I/O threads in LADS leads to contention in memory controller. To address this issue, we propose Multiple Memory Buffers (MMB), which distributes the RMA buffer across all the CPU sockets to reduce memory controller congestion.

(ii) The distributed RMA buffer at each CPU socket alone is not sufficient to improve the memory latency problem. In some cases, threads may try to access the remote RMA buffer hosted at different CPU sockets. To avoid such remote memory access, we design and implement Memory-aware Thread Scheduling (MTS) to schedule threads to access only the RMA buffer hosted on the same CPU socket. MTS reduces overall memory latency by eliminating all accesses to remote memory.

(iii) We conduct a comprehensive evaluation for our proposed ideas using a file size distribution based on a snapshot of the real peta-scale file system at ORNL [10]. We compare the performance of our proposed MMB and MTS with default settings where it uses single RMA buffer and where it applies NUMA binding to threads. From our experimental results, we have observed that our proposed idea yields up to 44% higher data transfer rate than the default settings.

The rest of this paper is organized as follows. Section 2 describes the LADS architecture and implementation details. Section 3 outlines the design and implementation of the proposed memory-level optimizations. Experimental setup and evaluation results are presented in Section 4. Section 5 describes the related works and we conclude in Section 6.

2. Layout-Aware Data Scheduling

Data sharing and scientific collaborations are advancing in recent years. Tools like GridFTP [4], bbcp [3], and LADS [1] are developed for efficient data transfer across geodistributed data storage facilities. LADS [1], an end-to-end data transfer tool, exploits the underlying storage architecture for optimizing the bulk data movement between data centers connected via high-speed terabit network. LADS uses Common Communication Interface (CCI) to fully utilize the terabit network [5] capabilities. The work proposed in this paper is an extension of LADS data transfer framework. This section describes LADS architecture.

Threads and Work Queues. LADS consists of three different types of threads, that is, Master (MT), Communication (CT), and I/O thread. The Master thread splits the workload in chunks and makes each chunk into a task. These tasks are inserted into the OST queue. In particular, there are as many OST queues as the number of OSTs in the Lustre file system. Master thread schedules I/O threads to OST queues and the I/O threads dequeue the tasks from OST queues to perform I/O operations. On the other hand, Communication thread manages the end-point communication between source and sink. The I/O thread loads the data chunks from storage to RMA buffer at source and stores them from RMA buffer to storage at sink. Both the Master and the Communication thread own work queues which hold the requests to transfer data objects to each other.
**Communication Protocol.** From now, the notation of source-end in the data transmission will be SRC, and the notation of sink-end in the data transmission will be SNK.

**Step 1.** The SRC MT catches the layout of data chunks for the requested file and adds the request to the SRC Communication Thread Work Queue (CWQ). The SRC CT sends a new file request to the sink-end via CCI end-point connection. Similarly, at the sink-end, SNK CT upon receiving the request forwards it to SNK Master Thread Work Queue (MWQ).

**Step 2.** The SNK MT upon receiving the request creates a new file with the same name as in the request. The file id corresponding to the newly created file is added to SNK CWQ in the request form and is sent to source-end. After receiving the request at source-end, SRC CT directs the request to SRC MWQ. The SRC MT loads the chunks information into SRC OST queues.

**Step 3.** Once the SRC MT inserts the data chunk into the SRC OST queue, SRC MT wakes up the SRC I/O threads according to the number of OSTs in the underlying SRC Lustre file system. Every SRC I/O thread at source-end then traverses the OST queues, gets the chunks information, reads the chunks from the physical OST storage, and loads them into the SRC RMA buffer. Then, SRC I/O thread inserts a request into the SRC CWQ to send the data chunk’s information. SNK CT receives the request, and it gets the data chunk from SRC RMA buffer via RDMA access.

**Step 4.** At SNK, after reading the data chunk from SRC RMA buffer, the SNK CT loads the data chunk into the SNK RMA buffer. Then the SNK I/O threads are scheduled to the SNK OST queues by the SNK CT to write the data chunk from the SNK RMA buffer to the SNK physical OST storage. The SNK CT sends the transfer completion message and repeats Steps 3 and 4, till all the data chunks of all the files are transferred.

### 3. Design and Implementation

This section describes the proposed optimization methods and its design and implementation details.

**3.1. Overview.** Figure 1 shows an overview of the proposed memory-level optimizations in LADS software framework [1]. The proposed approach consists of three major elements, (i) distributed RMA buffers at each CPU socket, (ii) Socket-based Memory-aware Thread Scheduling (SMTS), and (iii) NUMA node based Memory-aware Thread Scheduling (NMTS). The proposed optimizations are highly flexible and can be applied to \( n \) CPU sockets. Each CPU socket consists of cores, shared Last Level Cache (LLC), and one or more memory controllers, whereas the number of cores, LLC, and memory controller depend on the CPU type used. If there are multiple sets of cores and memory controllers in a CPU socket, then there will be more than one NUMA node per CPU socket. As shown in Figure 1, each CPU socket hosts three types of threads, that is, Master (MT), Communication (CT), and I/O thread; each thread performs a specific functionality similar to LADS architecture [1]. Proposed optimization methods are an extension of existing LADS implementation.

The Master thread captures the layout of files to be transferred from Lustre [8] object storage targets (OSTs) and schedules I/O threads to execute I/O operations specific to these OSTs. The I/O threads read the data chunks from the object storage into the RMA buffer and notify the Communication thread, which is responsible for creating an
end-point connection via CCI API [5] and sending the data chunks in RMA buffer from source side to sink side.

3.2. MMB: Multiple Memory Buffers. The LADS data transfer framework relies on single RMA buffer, where all the I/O threads load and store the data chunks between RMA buffer [1] and underlying storage system. However, the use of single RMA buffer can cause two problems: (i) memory latency caused by remote CPU socket’s memory access and (ii) contention on the memory controller. When threads are created, LADS schedules the threads to random CPU cores. Generally, I/O threads are scattered on all CPU sockets, that is, several I/O threads from different CPU sockets access a specific CPU socket’s RMA buffer. This incurs a number of remote memory accesses which have a longer latency than local memory access. Moreover, in LADS the number of I/O threads is configurable and in ideal scenario, it considers the number of OSTs belonging to the target Lustre file system for the number of I/O threads. But, in realistic scenarios, data centers using Lustre may exceed hundreds or thousands of OSTs, so the number of I/O threads will be configured according to the number of CPU cores. If a high number of I/O threads corresponding to the number of cores in a multicore environment access the single RMA buffer, the contention will occur in the memory controller of the CPU socket or NUMA node hosting the RMA buffer. We propose Multiple Memory Buffer (MMB) scheme, which distributes the RMA buffer to each CPU socket’s memory in the existing LADS framework. Figure 1 shows the RMA buffer distribution to each CPU socket. This partitioning of RMA buffer per CPU socket gives significant benefits. First, it reduces the memory controller contention significantly which is caused by single RMA buffer on increasing the number of I/O threads. Second, it reduces the memory latency due to the less number of remote memory accesses of I/O threads.

3.3. MTS: Memory-Aware Thread Scheduling. This section describes the details about the two levels of Memory-aware Thread Scheduling (MTS). First, we discuss the Socket-based Thread Scheduling and second, we present NUMA node based Thread Scheduling when CPU socket has multiple NUMA nodes.

3.3.1. Socket-Based Thread Scheduling. The partitioning of RMA buffer across each CPU socket gives the privilege to reduce the memory controller contention. But the remote memory access is still possible and it can increase memory access latency as compared to local memory access. To solve the remote memory access problem, we propose Socket-based Memory-aware Thread Scheduling (SMTS) scheme. SMTS schedules I/O threads between CPU sockets in such a way that all the I/O threads should access RMA buffer residing on the same CPU socket. Also, the RMA buffer is registered by the Communication thread and accessed via RDMA read/write operations through the end-point. So, a single and dedicated Communication thread is required per CPU socket to manage each RMA buffer. In our approach, each CPU socket has only one Master and Communication thread and configurable number of I/O threads. All threads and connections created per CPU socket are independent of other CPU sockets. The Master thread in each connection controls the chunk-level scheduling and transmits only the assigned files and I/O threads also work against the same objects designated by the Master thread. In this way, all I/O threads are pinned to their local CPU sockets’ cores.

3.3.2. NUMA Node Based Thread Scheduling. Here, we discuss the NUMA-aware optimizations required in LADS framework for CPU sockets equipped with multiple NUMA nodes. Considering the fact that it is possible to associate multiple NUMA nodes per CPU socket [11], it is highly necessary to schedule threads onto NUMA nodes at each CPU socket. To address this kind of cases, we propose NUMA node based Memory-aware Thread Scheduling (NMTS). NMTS is shown in Figure 1 as threads pinned at each NUMA node in every CPU socket. The motivation behind addressing thread scheduling at NUMA node is to avoid remote memory access inside a CPU socket when the socket is equipped with multiple NUMA nodes. To schedule threads inside a CPU socket, two elements are taken into account, (i) interaction between threads and (ii) fairness of core usage. In LADS [1] framework, to complete data transfer more efficiently, threads interact with each other at a high extent.

At first, Master and Communication threads interact, when data transfer begins. LADS framework has its own data structures for file metadata and keeps information about the file size, fd, and data chunk layout of the files required to transfer. Both the Master and Communication thread maintain work queues. In the rest of the paper, we will use MWQ to denote Master thread’s work queue and CWQ to represent Communication thread’s work queue. The Master thread schedules the requested file to CWQ at the start of data transfer. The Communication thread also sends a file request to the MWQ, when a file request is received from the sink side. In case of high transfer traffic, the interaction between Master and Communication threads increases. So, in such scenarios the placement of Master and Communication threads plays a major role. If Master and Communication threads are placed on different NUMA nodes, the performance can degrade due to remote memory access inside CPU socket.

Moreover, Master thread manages the information about OSTs containing data chunks in queue. The number of queues is equivalent to the number of OSTs in target Lustre file system. The I/O threads access these OST queues when loading data chunks from storage to RMA buffer to get information about the chunk. So, placing I/O threads near Master thread is also an important factor in performance optimization. However, due to the high number of I/O threads, it is not possible to keep all the I/O threads in same NUMA node with Master thread. Scheduling all the I/O threads on single NUMA node incurs core contention among I/O threads. Therefore, firstly we suggest scheduling Master and Communication threads on the same NUMA node and then placing I/O threads to NUMA nodes as close as possible to Master thread. We distribute I/O threads across the NUMA nodes to enhance the fairness of per core usage as shown.
in Figure 1. Figure 1 shows that Master and Communication threads are hosted by same NUMA node and I/O threads are hosted near to the Master thread whereby load is evenly distributed among the NUMA nodes’ cores.

4. Evaluation

4.1. Experimental Setup. In this experimental setup, we use a private testbed with eight nodes connected by InfiniBand (IB) EDR (100 Gb/s). The nodes use the IB network to communicate with each other. We use E5-2650 v4 with two CPU sockets (two NUMA nodes for each CPU socket and six cores for each NUMA node), 128 GB DRAM, running with CentOS 7.3, Linux kernel 3.10.0-514.21.1.el7.x86. Two of the eight nodes are used as data transfer nodes (DTN) for source and sink hosts. The other six nodes are used as two different storage systems. We built two storage systems for more realistic experiments.

(i) Testbed-I: when building a peta-scale storage system with Lustre file system, it uses hundreds of object storage servers with tens of object storage targets in each server. For example, Spider II is a center-wide, Lustre-based system for one of the fastest supercomputers in the world, Titan with two namespaces [1]. Each namespace has 144 OSSs, which manage seven OSTs and each OST is configured with 10 HDDs in RAID-10 [1]. That is, the I/O of the PFS used in realistic environment is too fast to use all the available bandwidth of the PFS. In order to emulate such environments, for each source and sink host, we use memory file system mounted on a server node via NFS v4.0 as a high performance file system.

(ii) Testbed-II: we experimented with a small-scale testbed equipped with Lustre file system for each source and sink host, which is configured using one OSS, one MDS, and eight OSTs, each mounted over 600 GB 10K RPM 6 Gbps SAS, 2.5” HotPlug HDD. For each file system, we created 8 logical volume drives on top of the HDDs to make each disk become an OST. We set stripe count to be 1 and stripe size to be 1 MB.

We developed an in-house memory benchmark program in C++ to measure the memory bandwidth between different NUMA nodes of source and sink hosts.

From Table 1, we can notice that in socket remote NUMA node is 4% slower when compared to in socket local NUMA node, whereas remote CPU socket NUMA node is 32% slower than in socket local NUMA node.

We use two representative file distributions to have two file groups appropriate for our small-scale testbed setup: one for small number of big files with 8 × 3 GB files, referred to as big-file workload, and the other for large number of small files with 6,000 × 1 MB files, referred to as small-file workload. HPC file size distribution follows a binomial distribution in terms of file system space occupancy and number of files: larger files occupy most of the file system space, but with fewer numbers of files. On the other hand, small files have a more number of files, but the file system space occupancy is very small [1].

For the convenience of analysis, we define the following schemes:

(i) Baseline: it uses a single RMA buffer. The RMA buffer has a different physical memory location depending on which NUMA node it is placed. Experiment is performed according to the position of RMA buffer and NUMA binding. In the experiment, Ni means that the single RMA buffer is located at NUMA node i, and NBi means that the single RMA buffer is located at NUMA node i with I/O threads bound on same node. If I/O thread count exceeds the number of cores, I/O threads are distributed to NUMA node i and its nearest node evenly.

(ii) MMB: the RMA buffer is partitioned by CPU socket. In our experiment, a single RMA buffer of baseline is partitioned into two.

(iii) MTS: it applies both Memory-aware Thread Scheduling Algorithms (SMTS and NMTS) to MMB.

For each iteration of memory file system and Lustre experiment, we cleared page caches of the source, sink, and storage servers for a fair evaluation.

4.2. Results

Evaluation of Memory Access Speed by Intimacy of Thread Type. In LADS, threads share data structures; for example, MT and CT share the MWQ and CWQ whereas I/O threads share OST queues with MT. RMA buffer is shared between CT and I/O threads. Thus, increasing the locality between threads and shared data structures (memory variables and queues) benefits to improve the performance.

In this experiment, we perform the evaluation according to the arrangement of MT, CT, and I/O threads on same and different CPU sockets.

Table 2 shows the throughput comparison for various placements of CT and MT on CPU sockets. The experiment was made using small files with Testbed-I. We can clearly see that placing CT and MT on the same NUMA node which has the RMA buffer helps in improving the data transfer rate. The
MTS is aware of thread types and prioritizes to schedule them on the same NUMA nodes. In the following evaluations, we run experiments by fixing the placement of the CT and MT on the same NUMA node with the RMA buffer in cases of NUMA binding.

**Evaluating MTS with High-Speed PFS.** To verify the effectiveness of the proposed MMB and MTS idea, we compare the data transmission throughput with baseline using a single RMA buffer. Particularly, we experiment by increasing the number of I/O threads to evaluate the performance according to CPU core utilization. Figures 2(a) and 2(b) show the comparison results for big and small file workloads, respectively, with Testbed-I. In order to confirm the performance limitation of the single RMA buffer and the maximum performance achievable by the NUMA binding, we evaluate the baseline by changing the NUMA node position of the single RMA buffer with and without the NUMA binding of threads. The RMA buffer locates on NUMA node $i$ in experiments $N_i$ and $NB_i$, and I/O threads are bounded with considering the location of RMA buffer in $NB_i$.

In Figure 2(a), we first analyze the results with the single RMA buffer (refer to results with labels in $N_i$ and $NB_i$). We can observe about 23% performance improvement on average with NUMA binding over without NUMA binding. As the number of I/O threads increases, overall performance improvement is observed till 8 I/O threads, whereas increasing I/O threads to 16, there is no significant improvement in the performance. This is due to the thread saturation of underlying file system. Moreover, we observe that performance is slightly lower in 16 I/O threads. We suspect the reason of performance degradation to be memory contention and remote memory access by the single RMA buffer.

Second, we compare the baseline results with MMB and MTS. From the results, we can confirm that overall performance improvement by MMB over the baseline varies from 13.0% to 34.7% and with MTS, its improvement is from 21.7% to 44%. That is, we observe that MTS further increases the performance of MMB by about 10%. Overall, data transfer rate increases as the number of I/O threads increases up to 16, but after that, improvement is not significant. We also see that, up to 2–4 I/O threads, MMB and MTS have little effect. When CPU cores are not fully utilized, single RMA buffer implementation outperforms partitioning the RMA buffer, because where performance storage is not fully exploited due to small number of I/O threads, dividing RMA buffer and binding threads to NUMA node incurs additional overhead. On the other hand, from 8 I/O threads, MMB and MTS show improved performance over the baseline. Particularly, in 16 I/O threads, we observe that the average performance improvements of MMB and MTS are 9.4–15.1% and 16.9–23%, respectively, over the baseline with NUMA binding. This shows that MMB and MTS contribute on the memory-level optimization in data transfer where storage is sufficiently fast.

In Figure 2(b), we have observations similar to the big file workload experiment. Up to 4 I/O threads, there is almost no performance difference. On the other hand, after 8 I/O threads, we can observe that MMB and MTS have on an average higher performance than baseline when we are over 16 threads. We observe little performance impact by MTS over MMB for small file workload, whereas MTS shows higher performance than MMB for big file workload.

**Evaluating MTS with Slow PFS.** If the storage performance is low, the performance improvement of MMB and MTS can be reduced. Thus, in the next experiment, we test with the source and sink hosts on which the small-scale Lustre file system is mounted (Testbed-II). Figure 3 shows the results of the experiment with big and small file workloads in Testbed-II. First, we observe that the maximum data rate is smaller than the experiment in Testbed-I. We see up to 1363 MB/s transfer rate with 8 I/O threads in big file workload and 435 MB/s

![Figure 2: Performance comparison for Memory-aware Thread Scheduling using Multiple Memory Buffers per CPU socket. The x-axis shows the I/O thread count. $N_i$ and $NB_i$ are for baseline experiments. The error bar depicts min. and max. deviation from the average of 3 iterations.](image)
rate in small file workload. Unfortunately, in Figure 3(a), we see that MMB and MTS have slight performance impact. In Figure 2(a), MMB and MTS improve the performance on 2000–3000 MB/s range of data transfer rate, but in small-scale Lustre file system which has just 1 OSS and 8 OSTs, MMB and MTS do not show any performance improvement. But, Figure 3(b) shows that in small file workload MMB and MTS depict performance improvement by 9.7% on average and up to 15.7% with 8 I/O threads. Because in baseline, only one pair of MT and CT sends 6,000 files metadata to other work queues sequentially, whereas in MMB and MTS, two pairs of MT and CT work on 3,000 files metadata simultaneously. It is confirmed that if the PFS I/O performance is low, it is difficult to take advantage of the NUMA effect. On the other hand, Testbed-I (memory-based network file system) represents the high-speed PFS environment and confirmed the effects of memory-level optimizations (MMB and MTS). Though the storage is not so fast that makes a bottleneck in end-to-end data transfer, dividing RMA buffer to each CPU socket increases the performance in large number of small files.

5. Related Work

The recent literatures addressing the high-speed data transfer tools include GridFTP [4], bbcp [3], and LADS [1]. bbcp transfers large amount of data efficiently using multiple TCP streams. However, bbcp uses single I/O thread and is unaware of the underlying file system layout. GridFTP [4] supports striping which allows data transmission to multiple peers when data is stored across a set of storage systems. But GridFTP does not consider the existing file system level data chunk layout of the files that are striped on the object-based parallel and distributed file systems. GridFTP and bbcp core design relies on file-based data transfer, whereas LADS [1] is a layout-aware object-based data scheduler which considers the storage layout and uses Common Communication Interface (CCI) [5] to exploit the high-speed network. Moreover, LADS implements a multithreaded architecture to benefit from parallel file system.

Along the line of advancements in storage and network, the CPU and memory performance is also improving by including the NUMA architecture. Multithreaded applications need to reflect such NUMA-awareness in their design to achieve the highest bandwidth from CPU and memory. Thread and memory locality in NUMA architecture plays a crucial role in performance. With benchmarking with STREAM triad, a research has shown that when benchmark allocates threads on a local NUMA node, the memory access is much higher than placing threads on a remote NUMA node through 1-hop and 2-hop [12]. Another study addressing NUMA-aware Thread Scheduling includes RAMSYS [2]. RAMSYS is a high-speed data transfer tool which uses dedicated threads in asynchronous fashion to incorporate pipelining in each step. But RAMSYS schedules threads without taking into account the underlying storage layout. Also, RAMSYS allocates single task queue on PFS and pushes all I/O requests to single task queue in a similar fashion to single storage device, whereas our approach uses multiple task queues as the number of OSTs in underlying Lustre file system, which enhances the performance on storage level. Our approach is an extension of existing LADS [1] architecture.

6. Conclusion

The advancement in storage, network, and CPU architecture has directed the existing data transfer software to align their design in order to achieve high throughput. Currently, the majority of data transfer tools such as GridFTP [4] and bbcp [3] do not support such design optimizations. LADS [1], a data transfer software designed for high-speed networks, considers the underlying storage architecture. However, LADS ignores the NUMA architecture. In this paper, we
propose a NUMA-aware resource and thread scheduling for optimized data transfer in high-speed network. Our approach involves three major sections, (i) distributed RMA buffer to each CPU socket, (ii) high-level Socket-based Thread Scheduling, and (iii) low-level NUMA node based Thread Scheduling. Our approach not only reduces the memory controller contention but also improves the memory access latency. The evaluation has shown improvement up to 44% for high performance file system when compared to baseline.

Disclosure

The portion of this work was presented as work in progress (1-page abstract) in the 2nd Joint International Workshop on Parallel Data Storage and Data-Intensive Scalable Computing Systems (PDSW-DISCS), held in conjunction with SC’17 [13].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean Government (MSIT) (no. 2015-0-00590, High Performance Big Data Analytic Platform Performance Acceleration Technologies Development). This work also used the resources of the Korea Institute of Science and Technology Information (KISTI), in Daedeok Science Town in Daejeon, South Korea. The authors thank Dr. Sungyong Park for his constructive comments that have significantly improved the paper.

References

Research Article

Deployment Strategy for Car-Sharing Depots by Clustering Urban Traffic Big Data Based on Affinity Propagation

Zhihan Liu, Yi Jia, and Xiaolu Zhu

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China

Correspondence should be addressed to Zhihan Liu; zhihan@bupt.edu.cn

Received 26 October 2017; Revised 19 January 2018; Accepted 11 February 2018; Published 15 March 2018

Academic Editor: Youngjae Kim

Copyright © 2018 Zhihan Liu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Car sharing is a type of car rental service, by which consumers rent cars for short periods of time, often charged by hours. The analysis of urban traffic big data is full of importance and significance to determine locations of depots for car-sharing system. Taxi OD (Origin-Destination) is a typical dataset of urban traffic. The volume of the data is extremely large so that traditional data processing applications do not work well. In this paper, an optimization method to determine the depot locations by clustering taxi OD points with AP (Affinity Propagation) clustering algorithm has been presented. By analyzing the characteristics of AP clustering algorithm, AP clustering has been optimized hierarchically based on administrative region segmentation. Considering sparse similarity matrix of taxi OD points, the input parameters of AP clustering have been adapted. In the case study, we choose the OD pairs information from Beijing’s taxi GPS trajectory data. The number and locations of depots are determined by clustering the OD points based on the optimization AP clustering. We describe experimental results of our approach and compare it with standard K-means method using quantitative and stationarity index. Experiments on the real datasets show that the proposed method for determining car-sharing depots has a superior performance.

1. Introduction

Big data exists everywhere and is providing with kinds of large data sets which can make people’s life more convenient and realize sustainable development [1]. Big data usually requires a set of techniques and technologies with new forms of integration to reveal insights from datasets that are diverse, complex, and of a massive scale [2]. Over the last few years, urban traffic data have been exploding, and we have truly entered the age of big data for transportation [3]. This situation inspires us to make some new attempts on urban traffic big data. In the paper, we propose a new attempt of urban traffic big data to determine the locations of car-sharing depots.

Car-sharing systems intend to offer an alternative model of car rental, by which users are permitted to use vehicles charged by hours [4]. These respects are used to evaluate car-sharing systems, namely, urban traffic environment, depots’ layout, and rental mode. As for the urban traffic environment, intuitively big cities are a good choice because they have high demand for public transportation. With regard to depot locations, car sharing is a most important long-term decision owing to the fact that it has a direct impact on quality, efficiency, and cost of service and affects profit and market competitiveness. Deploying car-sharing depots based on demand has been a big challenge due to the lack of realistic vehicle operational data. Therefore, a detailed study of determining depot locations precisely would be necessary.

To maximize profits by distributing the depots rationally, the following three respects should be considered. (1) Consumers aspect: the ideal distance of walking on foot is 0–3 km. If the distance is too long, the willingness of users to rent vehicles will decrease significantly. (2) Return on investment aspect: good locations of depots will significantly improve overall earnings. From existing car-sharing systems, we know that car sharing is overwhelmingly concentrated in metropolitan cores; around 95% of members are found in these settings [5]. For example, Autolib’ is a full electric car-sharing service in Paris. Up to July 2016, it offers over 1000 depots that can be found within a 5-minute walk in Paris. (3) Feasibility of depots construction aspect: generally speaking, car-sharing depots should be located in hot spots,
such as shopping malls, office building parking lot, and transport hub. These sites usually have enough available parking, and cost of construction is relatively low. On the whole, consumers hope they can rent vehicles as conveniently as possible. However, car-sharing service providers aspire to earn more and spend fewer on constructing depots at the same time. Considering these factors in an integrated manner, a frequently visited area by taxis is a good choice, namely, taxi hotspots. With widespread traffic sensors, urban traffic data is easily acquired and becomes of large scale. There are many methods to discover taxi hotspots from taxi GPS trajectory data. However, not all taxi hotspots are well suited for car-sharing depots. Having available parking spots is necessary for building car-sharing depots. Origin and Destination points (OD points) of users’ trips can be extracted from taxi GPS trajectory data, which reflect traffic hotspots and indicate the potential demand of car sharing. Based on the above theory, we propose a method to discover the traffic hotspots by clustering taxi OD points and determine the locations of car-sharing depots.

As there are so many clustering algorithms, different clustering algorithm gives different clusters. It is important to choose an appropriate clustering algorithm to make a balance between time cost and performance. One of the most popular clustering algorithms is K-means. However, K-means works well only when the number of clusters is known before clustering. It is exciting that another popular clustering algorithm, AP (Affinity Propagation) clustering algorithm, can determine the number of clusters spontaneously. Nevertheless, the complexity of AP is unacceptable, particularly when the dataset is of large scale. To improve the computing complexity of AP, this paper proposes an optimization method based on administrative region segmentation and sparse representation of similarity matrix. The results of this study demonstrate the benefit of large-scale data to determine the locations of car-sharing depots. The results can provide some guidance and suggestion to government and car-sharing service providers in the early stage of car-sharing system construction. Although this study only uses a specific city as a case, the proposed method and framework are also applicable to other cities.

The contributions of this paper mainly lie in the following two aspects:

(i) We propose a novel optimization approach to determine the depot locations by clustering massive OD points with AP algorithm based on administrative region segmentations. We propose a method based on density to optimize the parameter of AP and briefly introduce the principle and application range of AP clustering method for sparse similarity matrix.

(ii) We implement experiments on a large-scale dataset about containing ninety thousand OD points extracted from taxi GPS trajectories generated by about 12,000 taxis in Beijing. Our method produces about 50 points suited for the car-sharing depots. Then we evaluate our model with the net similarity between optimized AP and K-means. The results show that our AP has an advantage over K-means.

All the experiments show that our method is feasible and effective in determining car-sharing depots by clustering.

The remainder of the paper is organized as follows. Section 2 introduces some related works about locations of depots and taxi GPS data briefly. Section 3 presents the details of our method to determine the locations of car-sharing depots. Section 4 discusses the experimental results and analyzes the results. Conclusions and future work are discussed in Section 5.

2. Related Works

The majority of researches on determining the locations of car-sharing depots are dealing with urban traffic big data. In this section, we review some of the existing works.

2.1. Determining the Location of Depots. Urban big data enables a highly granular and longitudinal system, and it can help us understand city system and service better [6–10]. It can be used in many fields such as planning and governing cities, and business. For example, [3] applies big data to traffic flow prediction. Reference [11] presents a model to evaluate train timetable from the viewpoint of passengers’ data on rail transit lines. Reference [12] proposes a study about public electric vehicle charging stations using traffic big data.

Many kinds of urban traffic big data are used for the depot locations problem. Reference [13] presents an approach to optimize locations of depots in one-way car-sharing systems in which vehicle stock imbalance issues are solved by three trip selection schemes. Reference [14] presents a method to optimize the locations of bike sharing stations and the fleet dimension and measures the bicycle relocation activities required in a regular operation day. Reference [15] develops a simulation model that considers demand variability and one-vehicle relocation policy and tests the solutions provided by the previous MIP model. Reference [16] analyzes the performance of the car sharing service across all stations, estimates the key drivers of demand, and uses these drivers to identify future locations of depots. Reference [17] determines the locations of depots based on the predicted car-sharing demand. The basis and premise of determining the candidate depots have been given. However, it is still difficult to determine the candidate depots so that a detailed study is extremely necessary.

2.2. Analyzing Taxi GPS Trajectory Data. Taxi GPS trajectory data is an important and effective type of urban traffic big data for analyzing some certain problems about transportation. More and more researches begin to focus on taxi GPS data in recent years. There are a number of works on analyzing taxi GPS data. Reference [18] uses taxi GPS data to analyze traffic congestion changes around the Olympic games in Beijing. Reference [19] presents a method to construct landmarks-nodes graph. Landmarks are defined as frequently traversed road segments by taxis. They present an approach to split adaptively a day into different time segments based on the entropy and variance of the travel time between landmarks. This brings up an estimative distribution of the travel times.
between landmarks. Reference [20] proposes a method to construct a model of traffic density based on a large scale of taxi trips, which can be used to predict the traffic conditions and estimate the effect of emissions on the city’s air quality. Reference [21] develops a method to identify traffic hotspots based on taxis GPS data, which is based on the method of clustering taxi GPS data by $K$-means algorithm. However, the most obvious problem of $K$-means is that it needs an input parameter $K$, which means you must know how many traffic hotspots in advance. From above all, for depot locations problem, more attention to taxi GPS data analyzing is needed.

3. Methodology

This section focuses on introducing our method in detail, which aims at finding suitable locations and the number of car-sharing depots while satisfying consumers demand and minimizing the total cost. Assuming that the total demand of car-sharing depots is unknown but positively associated with taxi flows, our approach is to cluster taxi OD points and then find hotspots from continuous taxi GPS trajectories, which could be considered as the locations of car-sharing depots. Our architecture of this paper is shown in Figure 1. The framework consists of three major components: filtering raw data, clustering OD points, and the final exemplars filter. The detailed process will be introduced in the following sections.

3.1. Filtering Raw Data. Filtering the efficient points from taxi GPS trajectory data is the necessary preparation, because not all the trips are efficient. For example, some error data caused by the breakdown of the GPS-equipment or some invalid data cannot reflect the character of traffic flows validly. The location of car-sharing depots is determined by the travel demand of travelers. We just filter the origin and destination points of passengers’ trips which reflect the travel demand to some extent. And the OD points can be extracted from the continuous GPS trajectories according to trigger event.

Each taxi GPS point is described by a set of six elements: taxi id, trigger event, operation status, time, longitude, and latitude of GPS. “Taxi id” is the license of the car, which is a unique identifier for each taxi. “Trigger event” is the event that represents the taxi’s trigger status. When the trigger event is equal to 0, that means the taxi turns to the “no-load” status from others. And 1 means turning to “load,” 2 means “fortified,” 3 means “withdraw garrison.” “Operation status” is the operation status of the taxi. 0 means “no-load,” 1 means “load,” 2 means “Parking,” and 3 means “off-the-line.” “Time” is taxi’s current time (SGT), the format is “mm-dd-hh-mm-ss”. “Longitude” is the GPS coordinates of the taxi (East longitude and North latitude). For example, a data record \( (1143, 1, 1, 1106123843, 116.556101, 39.963646) \), it means the taxi 1143 was turning to “no-load” and the current time is 12:38:43 6th November, Beijing Time, and the taxi was located at 116.556101$^\circ$E and 39.963646$^\circ$N. To satisfy our demand, we need to filter points which trigger event has a sudden jump from 1 to 0 or 0 to 1, namely, OD points. It is worth mentioning that all the OD points are sorted by birth time.

3.2. Clustering the OD Points. In order to determine the locations of car-sharing depots, we make the cluster analysis on OD points based on AP clustering algorithm.

3.2.1. New Preference \( \{s(k, k)\} \) in AP Clustering. Firstly, we review the standard AP model [22]. For $N$ data points, the input is a set of pairwise similarities \( \{s_{ij}\} \), where $s_{ij}$ is the similarity of point $j$ to point $i$, and a set of exemplar preferences \( \{p_j\} \), where $p_j$ is the preference for choosing point $j$ as an exemplar. Generally, preference $p_j$ is set as the similarity $s_{ij}$ and influences the final number of identified exemplars. The goal is to select a subset of data points as exemplars and assign every nonexemplar points to the corresponding exemplar, so as to maximize the overall sum of similarities between points and their exemplars. There are two kinds of message exchanged between data points, namely,
responsible for how well point \(x_i\) serves as the exemplar of \(x_t\), \(a(i,k)\) reflects the evidence for how appropriate \(x_t\) choosing \(x_i\) as its exemplar. Equation (1) indicates that the update of \(r(i,k)\) decreases the similarity \(s(i,k)\) by removing the corresponding candidate exemplars from competition. Equation (2) represents the update process of \(a(i,k)\) and gathers evidence from data points as to whether each candidate exemplar would make a good exemplar.

The above update rules require only simple, local computations that are easily implemented, and messages need only be exchanged between pairs of points with known similarities. At any point during affinity propagation, availabilities and responsibilities can be combined to identify exemplars. For point \(i\), the value of \(k\) that maximizes \(a(i,k) + r(i,k)\) either identifies point \(i\) as an exemplar if \(k = i\), or identifies the data point that is the exemplar for point \(i\).

AP considers all data points as potential exemplars. It takes as input a set of similarity \(s(i,k)\), while \(s(k,k)\) is set by input preferences. In this paper, similarity is set to be negative Euclidean distance: for points \(x_i\) and \(x_k\), \(s(i,k) = -\|x_i - x_k\|^2\). Note that the preference can be used to control the number of final exemplars, with low preferences leading to small number of exemplars and high preferences leading to large number of exemplars. Generally, the preferences of all data points are set to be the median of the input similarities so that all data points are equally suitable as exemplars. That declares no prior inclination toward particular data points as exemplars.

However, it would lead some outliers to generate corresponding clusters which consist only relatively small data points in traditional AP algorithm. For example, Figure 2(a) shows the outliers in the AP clustering procedure, and the data points in the upper right corner would form a single cluster far from with the other clusters. However considering the scenario of this paper, outliers are the points where taxis seldom pass by. From the economic point of view, the outliers are not suitable to be individual candidate car-sharing depots. Therefore, we prefer to merge outliers into nearest high-density cluster. Based on the purpose, we propose a new formulation of the input preference as follows:

\[
s(k,k) = \frac{1}{N-1} \sum_{i=0,i\neq k}^{N} s(k,i), \quad \forall k \in N. \tag{3}
\]

The new preference \(s(k,k)\) is set to be the average of similarities between point \(x_k\) and others. This value is related to density around the point. The higher the density, the bigger the value; meanwhile the point is more like to be chosen as the exemplar. Figure 2(a) presents the cluster result which preferences are set to be the median of all the input similarities. We can see outlier \(X_{out}\) becomes a cluster consisting only itself. Figure 2(b) presents the cluster result in which preferences are set followed by (3). Obviously, \(X_{out}\) belongs to the nearest relative high-density cluster.

### 3.2.2. AP Clustering Based on Administrative Region Segmentation

It is intuitively plausible that AP's runtime is \(O(N^3)\) per iteration. However, as [23] presents, sharing computations allow us to compute messages efficiently which can reduce runtime within \(O(N^2)\) per iteration. Figure 3 presents the curve of runtime with the total number \(N\) ranging from 5000 to 30000 by 5000 per step. We can find that the runtime increases dramatically with \(N\) rising. Results of experiments confirm the conclusion. While the \(N\) is thirty
thousand, runtime is about 15 hours. If the \( N \) becomes larger, it is unacceptable. For example, the number of OD points within twenty-four hours is about one hundred thousand, and it may cost hundreds of hours.

In order to solve the above problem, we propose an optimized AP clustering method based on administrative region segmentation. Suppose that \( N \) OD points are distributed uniformly in \( R \) administrative regions. There are four following major steps of our method:

1. Allocate all OD points into different sets by administrative regions. The set of OD points in each region is named as AdminRegion\([x]\), \( x = 1, 2, \ldots, R \). Each region has nearly \( N/R \) OD points.

2. Implement standard AP in each AdminRegion\([x]\), and then we get a set of exemplars named AdminRegionCenter\([x]\) for each AdminRegion\([x]\). The time complexity of this step is \( O(RT(N/R)^2) = O(1/RTN^2) \), where \( T \) is the number of iterations.

3. Repeat the two steps above for every day’s OD points. Then we have each region’s exemplars in each day: AdminRegionCenter[day]\([x]\), day = 1, 2, \ldots, \( D \), \( x = 1, 2, \ldots, R \).

4. For each region, put all the exemplars of AdminRegionCenter[day]\([x]\) to a set, day = 1, 2, \ldots, \( D \). Implement AP on these sets separately. Then we have the final exemplars of each region.

For simplicity, we mark step (1) to (3) as stage 1 and step (4) as stage 2.

While AP based on administrative region segmentation works well, it still takes too much time due to the mass data. For example, the iterations of Haidian District in Beijing need 12 hours, which is obviously unacceptable. One improvement is that sparse similarity matrix to AP is applied. AP exchanges message between each point. If the similarity between two points is too low, the message between them is so few that we can set it to zero. In other words, we can set a minimal and acceptable similarity threshold \( \epsilon \), then the computational formula of similarity becomes as follows:

\[
s(i, j) = \begin{cases} 
  s(i, j) & \text{if} \ (s(i, j) > \epsilon) \\
  -\infty & \text{if} \ (s(i, j) < \epsilon).
\end{cases}
\]  

Then, we transform similarity matrix \( \{s(i, j)\} \) to sparse matrix. That means we just need to compute the nonempty elements. We can use triples to store the sparse matrix and calculate based on some techniques to reduce the complexity. For \( N \) OD points, \( \{s(i, j)\} \) have \( N^2 \) elements. Through above method, it becomes \( M \) elements. Obviously, \( M < N^2 \). The time complexity of step (2) can be decreased to \( O((1/R)TM) \) from \( O((1/R)TN^2) \). When the scale of dataset is quite large and the dataset is widely distributed, the method is obviously efficient.

Despite the fact that the sparse method can shorten the time, it also brings some problems. The similarity between two points involved in the calculation is limited in sparse threshold \( \epsilon \). The procedure of finding exemplars limited in a certain distance will increase the number of exemplars.

After above optimization procedure, AP clustering method based on administrative region segmentation now can be implemented to get a set of exemplars. These exemplars are selected from actual data points, informally called “centers.” It can be considered as a subset of representative points of all the points. So we take these exemplars as traffic hotspots. These traffic hotspots can be regarded as potential candidate depots.

Figure 4 shows clustered exemplars from thirty thousand OD points of Beijing. We find that 82 percent exemplars are located within the Fifth Ring Road. As we all know, the Fifth Ring Road is the boundary of densely populated area. At the macroscopic level, these cluster exemplars results conform to the reality very well.

3.3. Exemplars Filter. Based on the previous steps, we have a set of exemplars. However, we cannot simply regard all the exemplars as car-sharing depots. In some cases, some exemplars are so close that they almost overlap (Figure 4). What is more, the number of points in each cluster is different. Some exemplars can represent many points while others represent only a few. Therefore, we count the point number
of each cluster and rank it from the largest to the smallest. As shown in Figure 5, the $x$-axis is the indexes of exemplars, and the $y$-axis is the number of points, which have been clustered to the exemplar. We find a turning point in Figure 5, and it is about (24, 100). It means that only 24 exemplars have 100 or more attaching points. This interesting result is caused by the principle of AP. AP aspires to find an exemplar for each point, and it cannot remove outliers spontaneously. Considering the goal of this paper is to find high-density areas, we can only take the first 24 exemplars to make a brief analysis and evaluation.

Figure 6 shows part of the points in Terminal 3 Building of Beijing Capital International Airport before filtering. Figure 7 shows the points after filtering. Compared with Figure 6, points in Figure 7 are no longer overlapping. Figure 8 shows the positions of the first 24 exemplars. We can find out that the most of exemplars are hotspots in our life. It is especially sensitive to train stations, large-scale business, and residence districts.

4. Experiment and Results

In order to prevent noise interference and determine appropriate locations of car-sharing depots, we take a week’s taxi GPS data consisting of seven hundred thousand points in Beijing. Thus, the target of this section is to apply "administrative region segmentation" model to the large-scale dataset.

In the following sections, we describe the datasets used for the experiment in Section 4.1. Section 4.2 presents the details of the experiment based on the large-scale dataset, and finally we analyze the results in several respects.

4.1. Case Study Datasets. To determine the locations of car-sharing depots, we need to obtain two types of data, namely, OD points obtained from taxi GPS trajectory data and the boundaries of administrative regions.
**Table 1: The results of our method.**

<table>
<thead>
<tr>
<th>Region</th>
<th>N</th>
<th>1st stage $K$ (mean preference)</th>
<th>1st stage $K$ (new preference)</th>
<th>2nd stage $K$ (sparse AP)</th>
<th>2nd stage $K$ (standard AP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haidian</td>
<td>237820</td>
<td>21094</td>
<td>9125</td>
<td>4075</td>
<td>304</td>
</tr>
<tr>
<td>Chaoyang</td>
<td>206602</td>
<td>14457</td>
<td>9009</td>
<td>3355</td>
<td>268</td>
</tr>
<tr>
<td>Fengtai</td>
<td>176047</td>
<td>11534</td>
<td>6845</td>
<td>2625</td>
<td>219</td>
</tr>
<tr>
<td>Xicheng</td>
<td>100602</td>
<td>4890</td>
<td>3492</td>
<td>1190</td>
<td>159</td>
</tr>
<tr>
<td>Dongcheng</td>
<td>93261</td>
<td>4313</td>
<td>3134</td>
<td>1042</td>
<td>146</td>
</tr>
<tr>
<td>Shijingshan</td>
<td>40137</td>
<td>1848</td>
<td>1816</td>
<td>582</td>
<td>84</td>
</tr>
<tr>
<td>Daxing</td>
<td>15901</td>
<td>1612</td>
<td>1577</td>
<td>605</td>
<td>73</td>
</tr>
<tr>
<td>Shunyi</td>
<td>15385</td>
<td>1321</td>
<td>825</td>
<td>325</td>
<td>32</td>
</tr>
<tr>
<td>Tongzhou</td>
<td>4936</td>
<td>715</td>
<td>664</td>
<td>281</td>
<td>43</td>
</tr>
<tr>
<td>Mentougou</td>
<td>3185</td>
<td>304</td>
<td>271</td>
<td>97</td>
<td>25</td>
</tr>
<tr>
<td>Fangshan</td>
<td>1263</td>
<td>267</td>
<td>236</td>
<td>104</td>
<td>26</td>
</tr>
<tr>
<td>Sum</td>
<td>895139</td>
<td>62355</td>
<td>36994</td>
<td>14281</td>
<td>1379</td>
</tr>
</tbody>
</table>

(1) **Taxi GPS Trajectory Data.** The GPS trajectory data acquired in GPS-equipped vehicles represents the mobility patterns of the citywide human, from which we can get the origin and destination points of each taxi trip. The OD points can be extracted from continuous GPS trajectory data by trigger event.

In this paper, we utilized a GPS trajectory dataset generated by 12,000 taxis in Beijing from 5 to 11 November 2012. The GPS trajectory dataset consists of about seven hundred thousand GPS points. And we extracted 895,139 OD points from GPS trajectory dataset. We define the following notations for simplicity:

(i) $Q = \{1, 2, \ldots, q, \ldots, Q\}$: set of taxi GPS trajectory data
(ii) $N = \{1, 2, \ldots, n, \ldots, N\}$: set of OD points which are filtered out from $Q$
(iii) $K = \{1, 2, \ldots, k, \ldots, K\}$: set of locations of car-sharing depots

(2) **The Boundaries of Administrative Regions.** The boundaries of administrative regions are used to divide the OD points, which consist of polygon vertexes. We obtain it from Baidu map API (https://api.map.baidu.com/library/CityList/1.4/docs/symbols/BMapLib.CityList.html).

4.2. **The Results.** In this paper, OD points are of large scale and dense, and the span of space is large. It satisfies the specification and requirement of sparse AP completely. Therefore, we use the sparse AP in stage 1 of Section 3.2. Following the steps in Section 3, the results of different input preference and divided two stages are summarized in Table 1. Firstly, we focus on the comparison between different input preferences. The "mean preference" means that the input preference is the mean of all similarities in stage 1. The "new preference" means that the input preference of stage 1 is the expected value of similarities calculated by (4). We found that the number of clustering exemplars based on mean preference is larger than that based on new preference, because the new preference results in allocating the outliers into corresponding high-density areas and decreasing the total number $K$. Secondly, compared with the results of sparse AP and standard AP of stage 2, we find that the total number of sparse AP is larger than standard APs. The average elements in clusters of the sparse AP are 2.59, while the average elements in clusters of standard AP are 26.83. The former looks so bad because of incorrect use of sparse AP. We have mentioned that we use the sparse AP in stage 1, which causes the similarity between any two points of the input data in stage 2 and is limited over the sparse threshold. Therefore, in stage 2, it is difficult to find exemplars over the sparse threshold validly once more. When we adopt the standard AP in stage 2, the cluster results become better. Therefore, it is important to verify whether the dataset is suitable to use sparse AP. In summary, stage 1 with new preference and stage 2 with standard AP are the best choice.

For more information, we compare the results between AP and $K$-means in net similarity, which is defined as follows:

$$
net\text{Similarity} = \sum_k \left( s(k,k) + \sum_i s(i,k) \right).
$$

Net similarity measures the appropriateness degree of how exemplars explain the data. It is the objective function that AP and $K$-means try to maximize. We can use the net similarity to evaluate the performance of the clustering methods. As we can see in Figure 9, the net similarity of AP based on administrative region is a little larger than $K$-means in each region, especially in big regions such as Haidian District. This is due to sparse AP used in stage 1 is more suitable to large-area regions. Meanwhile in smaller regions, such as Xicheng District, our method does not show an outstanding advantage.

The results present that there are 1379 exemplars alternative to car-sharing depots. However, as we know, not all the exemplars are suitable, and we need to filter these exemplars manually. Following the method in Section 3.3, we analyze the number of points which should be clustered to the 1379
Table 2: Comparison with AP and K-means in time cost (min).

<table>
<thead>
<tr>
<th>N</th>
<th>K</th>
<th>K-means</th>
<th>AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>160</td>
<td>0.006457</td>
<td>3.503131</td>
</tr>
<tr>
<td>10000</td>
<td>262</td>
<td>0.016079</td>
<td>16.63446</td>
</tr>
<tr>
<td>15000</td>
<td>339</td>
<td>0.019477</td>
<td>33.17552</td>
</tr>
<tr>
<td>20000</td>
<td>402</td>
<td>0.027256</td>
<td>107.2591</td>
</tr>
<tr>
<td>25000</td>
<td>463</td>
<td>0.047055</td>
<td>335.4124</td>
</tr>
<tr>
<td>30000</td>
<td>535</td>
<td>0.059033</td>
<td>945.0208</td>
</tr>
</tbody>
</table>

Figure 9: Net similarity.

Figure 10: Count of number of exemplars’ attaching points.

Figure 11: Exemplars first 50.

eXemplars, respectively, as shown in Figure 10. Similarly, we find a turning point and its coordinate is about (43, 1500). Therefore, we take first 50 exemplars to make a brief analysis and evaluate. As Figure 11 presents, we can find that our method is sensitive to most of the traffic hotspots, especially to stations, airports, shopping centers, and hospitals. From the most direct sense, this result is as expected.

To evaluate the time cost of our AP and K-means method, we make another experiment with 5000 to 30000 OD points based on the steps above. The results of time cost are summarized in Table 2. Apparently, the time cost of AP far outweighs the K-means. It is noteworthy that the results showed in Line 3 time cost of K-means, which just executes for just once, where we set K as a result of AP. That is to say, we set the final cluster number of AP as the input K of K-means. In fact if there is no result of AP, we must determine the approximate K value through different attempts and experiments iteratively. It is hard to predict the range of K on the premise of massive data. However, the results do not convince that the time complexity of K-means performs better than AP. If we perform K-means iteratively to determine the approximate K, in the worst case scenario, the time complexity of K-means would be

\[
\sum_{k=1}^{N} O(T \ast N \ast K) = O\left(T \ast N \ast \frac{N(N + 1)}{2}\right)
\]

and meanwhile the time complexity of AP is \(O(N^2)\), which is less than K-means. Therefore, the combination of AP and K-means is the future direction of this paper to reduce the time complexity of K-means.

5. Conclusions

In this paper, we do a new trial on urban traffic big data about the determination of car-sharing depots. Experiments are implemented on the taxi GPS trajectory data in Beijing consisting of large-scale taxi OD points to study deployment strategy for car-sharing depots. To solve the highly complex problem caused by the large-scale data set, we present an optimization AP clustering method based on administration region segmentation. We define a new preference formula.
to solve the outliers problem in the process of clustering. In addition, we apply sparse AP on our method to decrease time cost. Combining theories and practices, we present the scope of application of our method. Meanwhile we have compared the objective function of AP and K-means in common, namely, net similarity. AP can not only overcome the biggest weakness of K-means, which is that K-means cannot determine K (the number of clusters) by itself, but also perform better in the net similarity. In spite of some measures taken to minimize the runtime of AP, it still takes too long. Thus, in practical applications, we can combine AP with K-means to achieve better performance. Although this study only takes Beijing as a case, the result of this paper indicates that this method has good versatility, because the method is not restricted to the data set itself. The methodological framework is applicable to any city only if the data set is available.

Owing to the fact that our method is simply based on clustering using taxi OD points, we have not considered much about whether the hotspots have the capacity to be car-sharing depots. Meanwhile, simply dividing dataset by regions is not rigorous. It may breaks some relevance especially in high-density areas. In this case, AP clustering based on grid segmentation is more reasonable. The grid segmentation means that taxi OD points are partitioned into multiple nonoverlapping grids to simplify representation of huge data points into smaller subsets. We plan to make further studies in these aspects.

**Conflicts of Interest**

The authors declare that there are no conflicts of interest regarding the publication of this paper.

**Acknowledgments**

This work is supported by the National Science and Technology Major Project (2016ZX03001025-003) and Special Fund for Beijing Common Construction Project.

**References**


Recent research and production environments are deploying more container technologies for the execution of HPC applications and for reproducing scientific workflows or computing environments. Research works, however, have not accounted for performance interference when executing corunning applications in containers though the absence of an efficient performance isolation layer cannot guarantee the absence of performance interference among multiple corunning applications which share resources. In this research, we propose an interference-aware scheduling method that mitigates the problem of performance interference based on applications’ I/O and CPU usage profiles. The proposed method estimates the amount of interference between various pairs of applications and coschedules them based on estimated interference. We evaluate the proposed method for both Bag-of-Tasks (BOT) scientific applications and scientific workflows and compare our method to the Weighted Mean Method. Our method improves the performance of the target scientific application by coscheduling applications with the least estimated interference ratios.

1. Introduction

Due to their almost zero start-up times, minimal runtime overhead, lightweight feature, and the relatively higher deployment density per physical host [1–3], Linux containers are being deployed as efficient virtual technology solutions in research and production environments and in the cloud [4, 5]. Scientific researchers, in particular, are deploying containers to reproduce scientific workflows [6] and benchmark Virtual Machines [7] and provide distributed storage [8]. Efficient resource allocation to the containers in a manner that guarantees high performance is thus of utmost importance. Correspondingly, performance evaluations of containers by Xavier et al. [9, 10] establish the need for an efficient isolation layer to facilitate resource sharing in HPC environments. From their experiments, Xen had better isolation than containers due to nonshared Operating System (OS) indicating potential performance interference when corunning applications in container-based virtualization systems.

Also, existing Container Cluster Managers (CCM) [11–13] allocate physical resources to containers in a manner that maximizes the deployment density per node [3] without considering the cost to performance. Ideally, there should be no interference in performance of corunning containers; however, due to contention for shared physical resources, the performance of a given application varies with the coexecuted applications on the same physical resource. Particularly, as some applications alternate between computation and I/O phases, performance interference is most apparent when corunning applications involve multiple I/O operations. Estimating the interference introduced by different applications concurrently running on a given node and scheduling applications accordingly has become a necessity when aiming to maximize the performance of applications. Research works so far have focused on the performance interference of data-intensive and I/O intensive applications in traditional hypervisor-based virtualization (HPV) environments; however, to the best of our knowledge, this research is the first to propose an interference-aware scheduling scheme for OS-level virtualization technologies.

The proposed method employs a two-phase scheduling strategy to mitigate interference in OS-level virtualized environments. Our proposed scheduler schedules BOT
scientifc applications and scientific workflows applications to resources based on the similarities in the CPU usage and I/O usage profiles to minimize the interference from coscheduled applications. In the first phase, it uses information on the CPU and I/O usage of applications to first estimate the interference between the different applications and then coschedules each application with another application of least interference. In the second phase, the applications are matched to container resources using different resource allocation strategies which would further mitigate interference. The proposed method makes the following novel contributions:

(i) It establishes the existence of performance interference during the execution of corunning scientific applications.

(ii) It introduces a two-phase interference-aware scheduler in OS-level virtualization environments to mitigate interference during the execution of applications in containers.

(iii) It estimates interference among applications based on computations using Euclidean distances.

(iv) It coschedules applications in a manner that minimizes contention for same physical host resources and hence minimizes interference.

The rest of this paper is organized as follows: Section 2 presents the related work and Section 3 describes our interference-aware scheduler. In Section 4, we explain our experiment environments and evaluate the results. We finally conclude this paper in Section 5.

2. Related Works

Interference-aware scheduling in virtualized environments is still a novel research area. Existing methods have investigated interference of Virtual Machines (VMs); however, there is little research on interference in OS-level technologies like Linux containers which are currently being deployed in many production environments.

In their investigations on resource and security isolation in container technologies through benchmark experiments, Soltesz et al. [14] demonstrate that container-based systems are twice more suitable for server-type workloads. This finding coupled with the findings of Xavier et al. [9, 10] establish the premise that, for HPC scientific environments, the absence of performance interference cannot be guaranteed.

In hypervisor-based environments, interference-aware schedulers such as DeJaVu [15] use an “interference index” to estimate interference between workloads. However, determining the interference index requires profiling of low-level metrics of workloads over many hours which is time-consuming. Also, workload clustering analysis is only used to determine which sets of applications require interference-aware scheduling without estimating the interference introduced by coscheduled applications.

TRACon [16] also predicts interference in paravirtualized environments based on the I/O usage of the guest and native driver domains. Interference prediction based on three models, Weighted Mean Method (WMM), the Linear Model (LM), and the Nonlinear Model (NLM) is applied and the application runtime and I/O throughput of the various applications are compared, respectively. With the results of their prediction, incoming tasks are coscheduled to resources with applications of least interference. However, this method was applied to only Virtual Machines (VMs). The proposed method employs a two-phase interference-aware scheduling method which first estimates the interference between applications using clustering analysis and then schedules the applications to suitable OS-level virtual resources in a manner that minimizes contention and hence mitigates interference.

The native Container Clustering Manager for the recently widely adopted Docker containers [17], Docker Swarm [11], employs a manager-agent deployment structure which includes a host that runs a Swarm manager and other hosts which run a Swarm agent each for the management of container clusters. The Swarm manager orchestrates and schedules containers on the hosts according to three (3) scheduling policies: bin-packing, spread, and random [18]. The proposed method adds interference awareness to the existing container placement strategies of Docker Swarm.

3. Interference-Aware Scheduling Based on Clustering Analysis

According to research [16, 19, 20], we consider interference as the change in the relative total execution time of an application due to the execution of concurrently running applications. We propose a two-phase interference-aware scheduling algorithm based on application clustering analysis in this section.

In the first phase of the proposed interference-aware scheduler, applications are coscheduled for execution according to the result of application clustering analysis using the $K$-means algorithm. After clustering the applications, the coscheduled applications are then scheduled to the suitable resources by a resource scheduler in the second phase of the proposed scheduling procedure. The interference-aware scheduling shows a scheduling procedure for multiple applications waiting to be scheduled onto container resources in a manner that mitigates interference.

First, using profile data of peak CPU and I/O usage, applications are clustered into specified number of clusters using the $K$-means algorithm [21]. Then, the interference between a selected application known as the target application and other applications in the clusters is computed according to an interference ratio. The application with the least interference is then coscheduled with the selected application and executed according to the container placement strategy indicated by the user. Each application is executed in a different container in all our experiments. The key notations used in this paper are listed in Notations.

3.1. Algorithm 1: Interference-Aware Scheduler. Algorithm 1 starts when a new application joins the queue with profile information on CPU utilization and I/O usage (Line (1)). The scheduler then uses the $K$-means clustering algorithm...
to group the applications into \( K \) clusters. The \( K \)-means algorithm partitions data into multiple sets with each containing a unique center or centroid. For a queue containing a set of applications, \( Q = \{x_1, x_2, \ldots, x_n\} \), with initial centroids given as \( C = \{c_i \mid i = 1, 2, \ldots, k\} \), the algorithm calculates the distance between each data point and cluster centroids.

Next, the scheduler randomly selects an application from any of the \( K \) clusters (Line (4)) known as the target application and calculates the interference ratio between that application and other applications in different clusters (Lines (5)-(9)). The scheduler then selects the application with which the target application has the least interference according to the calculated interference ratio (Lines (11)) and compares the interference ratio to a preset threshold value, \( \alpha \). If the ratio is less than the threshold value, the scheduler then invokes the resource selection method to select a suitable node from the cluster (Lines (13)-(14)).

The method returns nodes for the execution of each pair of applications into containers according to their resource demands. Then the applications are coscheduled to containers on the node accordingly. Otherwise, the system considers other placement strategies of Docker Swarm according to the clustering policies selected on the available nodes.

### 3.2. Interference Detection and Interference Ratio, \( r_{jh} \)

The proposed approach clusters all the applications in the queue into \( K \) clusters using profile data on their peak CPU and I/O usage. To calculate the interference ratio between the randomly selected application and other applications in other clusters, our approach computes a two-dimensional Euclidean distance between the selected application \( x_h(a, b) \) and the centroid of its cluster \( C_h(c, d) \) and that of other applications \( x_j(a, b) \) and the centroid of the target application \( C_j(c, d) \) according to (1). With the result, an interference ratio is calculated according to equation (2) and the results are used to select container resources for the execution of coscheduled applications.

\[
\text{CalcDist}(x_h, C_i) = \sqrt{(a - c)^2 - (b - d)^2}, \quad (1)
\]

\[
\text{Calc R} = \frac{\text{dist}(x_h, C_i)}{\text{dist}(x_j, C_i)}, \quad (2)
\]

### 3.3. Algorithm 2: Resource Selection Method

Interference-aware scheduling methods for scientific applications aim at minimizing the performance overheads introduced by coscheduling applications on a particular physical resource. It is therefore important that, in a container cluster, the resource that minimizes the likelihood of interference is selected. At the second phase of the proposed scheduling method, we adopt the placement strategies of Docker Swarm to schedule container resources to applications. The Swarm manager orchestrates and schedules containers on the hosts according to three (3) scheduling policies: bin-packing, spread, and random [18]. The bin-packing strategy chooses physical resources based on the highest number of containers running on that resource whilst the random scheduling strategy does not consider the number of running containers. The spread strategy, on the other hand, schedules applications to physical resources with the least number of containers running on them. We adopt the spread strategy as our default strategy with the assumption that the number of containers reflects the amount of resource contention for resources in the system.

When the resource selection method is called, the resource scheduler looks for available nodes and considers the container placement strategy indicated by the user. When the default strategy indicated is the spread strategy as used in our experiments, the node with the least number of containers is returned (Lines (4)-(5)). On the other hand, when the bin-packing strategy is used, the available node with the highest number of containers is returned (Lines (8)-(9)). With the random strategy, any available node is returned (Lines (11)-(12)). The method returns the results of executing the applications in the containers to the scheduling algorithm.

### 4. Experiments

Scientific applications are classified into Bag-of-Tasks (BOT) and scientific workflows [22] represented as Direct Acyclic Graphs (DAG). In this section, we describe the experiment environments and establish performance interference when corunning BOT applications and scientific workflows in OS-level virtualized environments. We then compare the proposed method to the Weighted Mean Method for BOT scientific applications. We also experiment which of the three (3) container placement strategies of Docker Swarm best mitigates performance interference for both BOT scientific applications and scientific workflows.

#### 4.1. Experiment Environments

We use Docker containers, a lightweight virtualization solution for fast creation and
Input: NodeList $N = \{(n_i) | i = 0, 1, \ldots, m\}$, Co-scheduled applications $[x_h(\min_c, \min_R)]$, 

1. Set resource available $== \text{true}$;
2. Set default $== \text{spread}$;
3. if resource available $== \text{true}$ then
   4. Node, $n_i \leftarrow \text{FindNode}(\min \text{cont})$;
   5. cont$_h \leftarrow \text{Create} [x_h(\min_c, \min_R)]$;
   else
   7. if default $== \text{binpack}$ then
      8. Node, $n_i \leftarrow \text{FindNode}(\max \text{cont})$;
      9. cont$_h \leftarrow \text{Create} [x_h(\min_c, \min_R)]$;
      else
      11. Node, $n_i \leftarrow \text{FindNode}(\text{ran cont})$;
      12. cont$_h \leftarrow \text{Create} [x_h(\min_c, \min_R)]$;
      end if
  end if
14. end if
15. Results $\leftarrow \text{Execute}(x_h)$;

Output: Execution Results

Algorithm 2: Resource selection method.

Table 1: Profiles of applications characteristics.

<table>
<thead>
<tr>
<th>Application</th>
<th>I/O Read (KB)</th>
<th>I/O Write (KB)</th>
<th>CPU usage%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update-Heavy</td>
<td>2,600</td>
<td>55</td>
<td>97.4</td>
</tr>
<tr>
<td>Read-Only</td>
<td>1,890</td>
<td>1</td>
<td>82.2</td>
</tr>
<tr>
<td>Read-Modify-Update</td>
<td>1,150</td>
<td>10</td>
<td>40.6</td>
</tr>
<tr>
<td>Melt 3500</td>
<td>44</td>
<td>18</td>
<td>100</td>
</tr>
</tbody>
</table>

4.2. Performance Interference in Bag-of-Tasks (BOT) Scientific Applications. We first validate the postulation that there is performance interference in containers when corunning BOT scientific applications using an illustrative experiment. For this experiment, we consider scientific applications with four different workloads. Three (3) of the workloads are variations of YCSB applications, each with a different I/O and CPU usage profile, whilst the other application is Melt, a variation of LAMMPS [23]. The workloads and their I/O characteristics are briefly described as follows.

Yahoo Cloud System Benchmark (YCSB) [24], a basic benchmark for cloud systems, includes a set of six (6) core workloads that define the running of data-intensive applications in virtual environments. These benchmarks represent applications with varying I/O intensities and I/O access patterns. For these experiments, we deploy three (3) of the workloads, the Update-Heavy, Read-Only, and Read-Modify-Write workloads, due to their various I/O intensities and I/O access patterns. The I/O access pattern of Update-Heavy Workload and Read-Only Workloads take the form of random reads whilst the Read-Modify-Write workloads access I/O resources in a bursty manner with maximum I/O reads taking place at the beginning and end of the executions.

Melt, a variation of a molecular dynamics code known as Large-scale Atomic Molecular Massively Parallel Simulator (LAMMPS) [23], is used for simulating different types of particle behaviors. Melt simulates the rapid melting of a 3d LJ system and accesses I/O resources in a bursty manner with maximum I/O reads taking place at the beginning and end of the executions.

Using Glances profiling tool [25], we obtain data of the I/O and CPU usage by each of the above scientific applications when run alone as shown in Table 1. For each characteristic, the data represented is based on the peak value obtained for each application during their execution lifecycle. We evaluate only two characteristics, CPU and I/O, per application and perform clustering analysis based on them.

In the experiment, we select the Update-Heavy workload as our target application and assume that only two containers can run on the same node simultaneously. Thus in this experiment scenario, we coexecute the Update-Heavy workload with Read-Only workload, YCSB Read-Modify-Update workload, Melt (3500), and another YCSB Update-Heavy workloads, respectively, under the same conditions.
We present the results of the change in execution times of the Update-Heavy workload when co-run with other workloads as well as the change in throughput in Figure 1.

Figure 1 shows the changes in execution time of Update-Heavy workload from Cluster 1, when coexecuted with different applications. The execution time of YCSB Update-Heavy in each run is compared to the execution time of Update-Heavy when executed alone in a container on the physical node. In this experiment, the change in the execution time is reflective of the amount of performance interference due to coscheduling with other applications.

From the results, YCSB Update-Heavy workload with Read-Only update workload relatively experiences the most interference of about 293 seconds when coexecuted whilst YCSB Update-Heavy workload experiences the least interference of 34 seconds when coexecuted with Melt application. We attribute these variations in the execution times to contention for same I/O resources. Particularly, YCSB Read-Only workloads take a longer time to process data requests and retrieve the required data from the database, thus increasing the time spent on I/O operations. In effect, the YCSB Update-Heavy workload spends more time contending for I/O resources shared with YCSB Read-Only workloads. Consequently, the throughput for YCSB Update-Heavy workload when coexecuted with Read-Only workloads is the least due to relatively longer execution time. This validates our hypothesis that there is performance interference among containers on the same physical resource. The results also show that it is imperative to estimate the performance interference between colocated applications during the coscheduling of applications.

### 4.3. Comparing the Proposed Method and the Weighted Mean Method (WMM)

We evaluate the relationship between the proposed interference ratio and the execution times of some Bag-of-Tasks applications and compare our results with results obtained from scheduling using Weighted Mean Method and a noninterference-aware method based on the bin-packing scheduling method of Docker Swarm.

Weighted Mean Method [16] also uses Euclidean distances to determine weights of each application according to the Principal Component Analysis (PCA). Some research works [16, 26] use the similarity between workload characteristic vectors of applications and account for noise, redundancy, and similarity in applications characteristics by using the principal components of a vector. It then uses the reciprocal of their distances as the weights to get the predicted response.

In Table 2, we present the details of the estimated interference calculated using both the proposed method and the Weighted Mean Method. Both estimations are based on the CPU usage and I/O of four (4) workloads of scientific applications when executed with different parameters. We choose Update-Heavy workload as our target application and conduct the ensuing experiments with it. The table contains the interference ratio of all the applications in the queue waiting to be scheduled to resources in the system. From the results, applications in Cluster 0 have the highest amount of interference with Update-Heavy workload.

In this experiment, we submit 1 Update-Heavy workload, 1 Melt application, and 1 Read-Only workload and coexecute them in three runs using three scheduling strategies: the proposed interference-aware method, Weighted Mean Method, and no-interference scheduling methods. The no-interference method, which does not account for interference, is based on Docker Swarm’s bin-packing strategy and chooses resources based on the highest number of containers running on that resource.

In each run, the proposed method schedules both Update-Heavy workload and Melt on the same node because Melt has the least interference of 0.00015 compared to 0.00052 when scheduled with Read-Only workload. The Weighted Mean Method however schedules both Update-Heavy workload and Read-Only workload on the same node because Read-Only workload has the least interference of 0.43790 compared to 1.48595 when scheduled with Melt. The no-interference scheduling method which does not consider

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Application</th>
<th>I</th>
<th>WMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 0</td>
<td>Update-Heavy</td>
<td>1.00000</td>
<td>0.00023</td>
</tr>
<tr>
<td>Cluster 0</td>
<td>Read-Only</td>
<td>0.00052</td>
<td>0.43790</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Read-Modify-Update</td>
<td>0.00267</td>
<td>0.85731</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Melt</td>
<td>0.00015</td>
<td>1.48595</td>
</tr>
</tbody>
</table>
interference ratio between applications also coruns Update-Heavy workload with Read-Only workload which has a relatively higher interference ratio of 0.00052 according to our proposed method.

Figure 2 describes the scheduling result showing execution times of YCSB Update-Heavy workload when coexecuted with other applications using different scheduling strategies. For each scheduling method, we obtain the execution time of the workflow, YCSB Update-Heavy workload, and compare the results. From the results, the execution time of our proposed method is least at 732.89 seconds. However, the execution time of the no-interference and WMM is same at 1003.45 seconds since they coexecute the same applications.

The WMM model predicts interference by calculating the Euclidean distances between the data points in the clustering space and chooses three nearest data points whose reciprocal is used as the weights to get the predicted response. This does not guarantee an accurate prediction when the applications to be scheduled have highly variable differences in I/O intensities. The proposed method, however, guarantees a more accurate relationship between the prediction and the actual performance interference of applications because it predicts interference based on the distances between the target application and the application considered for coexecution.

4.4. Interference-Aware Scheduling for Dynamic Workloads. In the previous scenarios, the proposed method is able to improve the execution time for which the Update-Heavy workload application is executed for static workloads. In other words, all the applications were present in the queue at the time of the scheduling. In this section, however, we examine the scheduling of a dynamic workload since in real scenarios, tasks arrive dynamically. Thus, a simulation experiment using CloudSim [27], which considers the arrival of different tasks every 15 seconds, is conducted in this section to show the improvement in throughput for both Heavy I/O tasks and light I/O tasks by the proposed interference-aware method. We modify cloudlets to have Heavy I/O tasks and light I/O tasks according to their use of resources.

We begin the execution with 50000 tasks and increasingly submit tasks at an interval of 15 seconds. We increase the job size from 50000 tasks to 100000 tasks, 150000 tasks, and 200000 tasks for both types of tasks. As soon as the jobs arrive, interference is calculated and the applications are rescheduled according to the interference-aware method. This translates to an increased number of tasks being executed within a given period of time since tasks are scheduled in a manner that reduces interference.

Accordingly, the throughput of all the applications in the system is relatively higher than for jobs which are scheduled with the no-interference-aware method:

\[
\text{Throughput}, T = \frac{\text{Number of applications}}{\text{Execution time}}. \tag{3}
\]

The threshold for the interference-aware method is randomly set at 0.2 and throughput \(T\) is calculated according to (3) as the number of tasks completed during an execution, per the time of execution. As jobs with different characteristics are executed, their throughput and the number of jobs executed are compared using the proposed method and the no-interference scheduling method.

From the results in Figure 3, the execution of tasks using both the proposed method and a no-interference method decreases in throughput for increasing number of applications due to increasing time taken to execute the applications as a result of interference and an increasing number of tasks which were not scheduled for execution due to the set interference ratio. However the throughput of the proposed method is slightly higher due to the reduced time taken for execution as a result of I/O interference mitigation. Also, Heavy I/O tasks for both scheduling methods have a lower throughput since most of the execution time of these tasks are spent performing I/O operations instead of computations. This adds to the fact that including interference awareness in scheduling decisions helps improve the throughput and the performance of applications.

**Figure 2:** YCSB Update-Heavy with different scheduling strategies.

**Figure 3:** Throughput for tasks using interference awareness.
Table 3: Applications’ profiles, clusters, and their interference ratios.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Application</th>
<th>Memory usage (MB)</th>
<th>CPU usage%</th>
<th>Interference ratio, ( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 0</td>
<td>CFD 512 KB</td>
<td>403,230,720</td>
<td>206.7</td>
<td>0.000432</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Melt 500</td>
<td>23,183,360</td>
<td>100</td>
<td>0.0194</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Melt 1000</td>
<td>23,314,432</td>
<td>100</td>
<td>0.0191</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Melt 1500</td>
<td>23,445,504</td>
<td>100</td>
<td>0.01886</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Melt 2000</td>
<td>23,576,576</td>
<td>100</td>
<td>0.01858</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Melt 2500</td>
<td>23,707,648</td>
<td>100</td>
<td>0.01831</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Melt 3000</td>
<td>23,838,720</td>
<td>100</td>
<td>0.0186</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Melt 3500</td>
<td>23,969,792</td>
<td>100</td>
<td>0.01781</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Peptide 500</td>
<td>32,460,800</td>
<td>100</td>
<td>0.0093</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Peptide 1000</td>
<td>33,382,400</td>
<td>99.99</td>
<td>0.0089</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Peptide 1500</td>
<td>34,213,888</td>
<td>99.99</td>
<td>0.0085</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Peptide 2000</td>
<td>35,606,528</td>
<td>100</td>
<td>0.00797</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Peptide 2500</td>
<td>36,810,752</td>
<td>100</td>
<td>0.00754</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Peptide 3000</td>
<td>37,613,568</td>
<td>100</td>
<td>0.00727</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>Peptide 3500</td>
<td>38,817,792</td>
<td>100</td>
<td>0.00691</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>GALFA 5</td>
<td>14,708,736</td>
<td>132.9</td>
<td>1</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>GALFA 10</td>
<td>14,716,928</td>
<td>126.2</td>
<td>0.9535</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>GALFA 15</td>
<td>14,745,600</td>
<td>121.01</td>
<td>0.8199</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>GALFA 20</td>
<td>14,766,080</td>
<td>119.5</td>
<td>0.745</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>GALFA 25</td>
<td>14,782,464</td>
<td>117.9</td>
<td>0.6949</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>CFD 2 KB</td>
<td>13,967,360</td>
<td>201.4</td>
<td>0.2928</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>CFD 32 KB</td>
<td>14,098,432</td>
<td>205.2</td>
<td>0.3796</td>
</tr>
</tbody>
</table>

4.5. Performance Interference for Scientific Workflow Applications. We also validate interference awareness for coexecuting scientific workflows using similarities between CPU and memory usage profiles. For this experiment, we select 1 scientific workflow, Montage GALFA [28], and three (3) BOT applications, CFD [29], Melt, and Peptide [23]. We cluster these applications into three clusters based on the CPU and memory usage profiles when the applications are coexecuted with Montage GALFA as shown in Table 3.

Montage GALFA, a data-intensive workflow application, is an Astronomical Image Mosaic Engine for creating mosaics using multiple astronomical images. In this paper, the Montage GALFA application shrinks five (5) data cubes by averaging different number of planes (5, 10, 15, 20, 25) and then aggregates them into a mosaic [30] following three major steps.

The second target application is an aerodynamic variation of a Computational Fluid Dynamics (CFD) simulation application used for 2-dimensional Euler unsteady flow analysis. In this experiment, we deploy different meshes with sizes 2 KB, 32 KB, and 512 KB, respectively.

The third and fourth target applications, Melt and Peptide, are variations of LAMMPS. Whilst Melt simulates the rapid melting of a 3d LJ system, Peptide simulates the granular particle pour and flow of both 2D and 3D systems. We iteratively run coscheduled applications which have shorter execution times such as Melt and Peptide applications throughout the experiments to maintain fairness.

Table 3 describes the result of performing clustering analysis for the applications using the proposed approach. From the analysis, the applications are grouped into 3 clusters. Cluster 1 has the highest number of applications (15) whilst Cluster 2 has 7 applications and Cluster 0 has only 1 application, respectively. We choose Montage GALFA (5) as our target application from Cluster 2 and conduct the ensuing experiments with it. The table also contains the interference ratio of all the applications in the queue waiting to be scheduled to resources in the system. From the results, applications in Cluster 2 have the highest amount of estimated interference with Montage GALFA (5 planes) using the proposed method.

Figure 4 shows the changes in execution time of GALFA workflow application, in Cluster 2, when coexecuted with applications from different clusters, \( C_0 \), \( C_1 \), and \( C_2 \), respectively. The interference ratio of the applications relative to \( C_2 \) is 0.00043, 0.01781, and 1, for \( C_0 \), \( C_1 \), and \( C_2 \), respectively. The execution time of \( C_2 \) in each run is compared to the execution time of \( C_2 \) when executed alone in a container on the physical node. In this experiment, the difference in the execution time is considered as the amount of performance interference due to coscheduling with another application.

From the results, \( C_2 \) experiences the most interference of about 63 seconds when executed with a similar application from the same cluster whilst \( C_2 \) experiences the least interference of 4 seconds when coexecuted with \( C_0 \). This also validates our hypothesis that there is performance interference among containers on the same physical resource and proves that our proposed interference ratio accurately predicts the interference among applications from different clusters. The results also show that it is imperative to
clearly understand the performance relationship between colocated applications during the coscheduling of applications.

4.6. Performance Interference for Montage GALFA Using Different Scheduling Strategies. In this experiment, we submit 1 Montage GALFA (5 planes) workflow, Melt (3500) application, and 1 Peptide (3000) application and coexecute them in three runs using three scheduling strategies: bin-packing, random, and the proposed interference-aware scheduler. The bin-packing strategy chooses resources based on the highest number of containers running on that resource and so schedules all the applications on the same node. The random scheduler and the proposed method, however, schedule both Montage GALFA (5 planes) and Peptide (3000) on the same node. The proposed method considers the interference ratio between the applications in making the scheduling decision and chooses Peptide (3000) which has a ratio of 0.00728 relative to 0.0181 of Melt (3500) applications. Due to the differences in execution times of both applications, Melt (3500) and Peptide (3000) applications are run iteratively in 20 runs to cover the span of time for which Montage GALFA (5 planes) is executed.

Figure 5 describes the scheduling result of our interference-aware scheduling algorithm, as shown in Section 3. For each scheduling method, we obtain the execution time of the workflow, Montage GALFA (5 planes), and compare the results. From the results, the execution time of our proposed interference-aware method and the random method is the least at 2117 seconds whilst that of bin-packing strategy is the highest at 2177 seconds. This is because the bin-packing strategy schedules all the applications on the same node without considering the contention for shared resources. Also from Figure 5, our method is able to improve the throughput of Montage GALFA application by 4.58% relative to the throughput when executed alone. This also shows that our method is able to mitigate interference.

5. Conclusions

In this paper, we propose an interference-aware scheduling method for both BOT and scientific workflows in different experiments. For BOT applications, the proposed method is compared with the Weighted Mean Method and evaluated for dynamic workloads whilst for scientific workflows the methods efficiency is evaluated for three different container placement strategies. From each of our experiments, we establish the fact that there is performance interference among applications when coexecuted using containers on the same node. Our proposed method however reduces the amount of interference when compared with other methods.

In the future we further investigate different interference prediction approaches for applications run in OS-level virtualized environments.

Notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>A queue with a set of applications waiting to be scheduled</td>
</tr>
<tr>
<td>𝑥ₖ</td>
<td>An application in a queue Q with profile data on resource usage and an application in Cluster(𝑖)</td>
</tr>
<tr>
<td>Cluster(𝑖)(𝑥ₖ, 𝑐ᵢ)</td>
<td>An 𝑖th Cluster containing 𝑥ₖ data points and centroid 𝑐ᵢ</td>
</tr>
<tr>
<td>𝑥ₗ</td>
<td>Another data point in Cluster(𝑖) selected for coexecution with 𝑥ₖ</td>
</tr>
<tr>
<td>𝑐ᵢ</td>
<td>The centroid of Cluster(𝑖)</td>
</tr>
<tr>
<td>𝑟ₗ,𝑗</td>
<td>The interference ratio between 𝑥ₗ and 𝑥ₖ</td>
</tr>
<tr>
<td>SelectXₗ(𝑥ₖ, 𝑐ᵢ)</td>
<td>Randomly selecting an application from a cluster</td>
</tr>
<tr>
<td>CalcDist(𝑥, 𝑐)</td>
<td>Calculating the distance between data point and centroid</td>
</tr>
<tr>
<td>CalcR(𝑥, 𝑐)</td>
<td>Calculating interference between two points</td>
</tr>
<tr>
<td>SelectMin(𝑟ₗ,𝑗)</td>
<td>Selecting the application with the least interference ratio</td>
</tr>
<tr>
<td>𝛼</td>
<td>A threshold value for interference</td>
</tr>
</tbody>
</table>
default: Container placement strategy indicated by the user
\(n_i\): A list of nodes in the node cluster
\(n_i\): An \(i\)th node from the list of nodes in the node cluster
\(c(n_i)\): An \(i\)th container on node \(n(i)\)
FindNode(): Returns available node according to the parameter indicated
\(\min_{cont}\): Node with minimum number of containers
\(\max_{cont}\): Node with maximum number of containers
\(\text{ran}_{cont}\): Randomly selected available node.

Conflicts of Interest
The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments
This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (Ministry of Science and ICT) (NRF-2015M3C4A7065646).

References
Research Article

Cultural Distance-Aware Service Recommendation Approach in Mobile Edge Computing

Yan Li1 and Yan Guo2

1School of Business and Management, Shanghai International Studies University, Shanghai, China
2State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China

Correspondence should be addressed to Yan Guo; guoyan@bupt.edu.cn

Received 13 October 2017; Accepted 26 December 2017; Published 14 February 2018

Aca
demic Editor: Youngjae Kim

Copyright © 2018 Yan Li and Yan Guo. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In the era of big data, traditional computing systems and paradigms are not efficient and even difficult to use. For high performance big data processing, mobile edge computing is emerging as a complement framework of cloud computing. In this new computing architecture, services are provided within a close proximity of mobile users by servers at the edge of network. Traditional collaborative filtering recommendation approach only focuses on the similarity extracted from the rating data, which may lead to an inaccuracy expression of user preference. In this paper, we propose a cultural distance-aware service recommendation approach which focuses on not only the similarity but also the local characteristics and preference of users. Our approach employs the cultural distance to express the user preference and combines it with similarity to predict the user ratings and recommend the services with higher rating. In addition, considering the extreme sparsity of the rating data, missing rating prediction based on collaboration filtering is introduced in our approach. The experimental results based on real-world datasets show that our approach outperforms the traditional recommendation approaches in terms of the reliability of recommendation.

1. Introduction

With the rapid development of Internet of Things and 5G network, a large number of distributed data and computation tasks are generated, which makes traditional centralized computing paradigm suffer large processing pressure. Mobile edge computing is emerging to process the data at the edge of network for high performance big data processing [1].

In mobile edge computing, services are provided within a close proximity of mobile users by servers at the edge of network [2]. With the increasing number of edge services, there are too much choices for users to meet their requirements [3, 4]; therefore, the service recommendation technology is needed to help people find the optimal edge services from a huge mass of services. Because of the ubiquitous services, service recommendation is playing an increasingly significant role in our diary life [5, 6]; for example, Amazon has deployed its recommending system to help recommending books and other products to its users [7].

Although there is a significant research in mobile edge computing, there has been little attention paid to services recommendation in mobile edge computing. Traditional service recommendation system is to dig up service users’ preferences based on user’s history records and scores and then recommend similar users’ services which have never been used by the service user [8, 9]. It can be reduced to a problem of estimating ratings for the services that have not been used by a user. Intuitively, this estimation is usually based on the ratings given by the user to other services. Once the ratings for the yet unrated services can be estimated, we can recommend to the service user the service with the highest estimated rating.

Much research over the past decade has focused on developing service recommendation systems, and they are usually classified based on recommendation approach as content-based recommendation [10], collaborative filtering-based recommendation [11], and model-based collaborative filtering recommendation [12].

Collaborative filtering is one of the most successful recommendation technologies in many recommended systems [13]. The basic idea is to predict the ratings of the services that have not been used by the service user according to the users
who have the common experience and the same interest or the services which are similar to the services that have been rated by the service user [14]. However, the efficiency and accuracy of collaborative filtering are not high in practical applications [15].

Traditional collaborative filtering recommendation approach only focuses on the similarity extracted from the rating data in the calculation of similarity, the selection of the nearest neighbor, and recommendation. Specifically, the recommendation is according to the weighted average of evaluation from the nearest neighbor, and the weight is the similarity between the user and the nearest neighbors. However, the similarity is not the only decision factor; there are many other factors having an important role. The preference of users only depends on the similarity, which may lead to an inaccuracy expression of user preference and low reliability of recommendation results.

In addition, traditional service recommendation approaches perform poorly when facing sparse user rating data. There are some approaches that have been proposed to solve the sparsity of ratings data, and it can be divided into three categories. The first is attribution and dimension reduction; however, attribution may lead to the missing of user's personal characteristics data, and dimension reduction may delete useful data. The second effective way to solve the sparse data problem is to introduce the trust into the recommendation system [16, 17]. The third way is to predict the missing ratings by adopting the approach in the similarity calculation [18]. This approach can exploit the effective data while also enlarging the influence of the erroneous data.

Considering that there is a close relationship between the user preference and their cultural background, we introduce cultural distance which can express the user preference to reduce the decision-making power of similarity. For example, in mobile edge computing, users in different edge clouds may have different preference, and users in the same edge clouds may have the same preference.

Responding to the above problems, we propose a new collaborative filtering recommendation approach based on cultural distance in which we combine the cultural distance and similarity to represent the user preference. In the approach, we combine the user-based collaborative filtering and the service-based collaborative filtering approach to take full advantage of the information of the similar users and similar services, and then we recommend service to users based on a cultural distance-aware collaborative filtering recommendation approach. In addition, we predict the missing ratings based on the cultural distance-aware collaborative filtering approach to improve the sparsity of user rating data.

The remainder of the paper is organized as follows: Section 2 introduces the background of service recommendation; Section 3 describes our cultural distance-aware hybrid collaborative filtering recommendation approach; Section 4 presents the experiment results and Section 5 concludes this paper.

2. Background

2.1. Related Concept. The main idea of our approach is the introduction of cultural distance; therefore, we introduce a related concept: cultural distance before the introduction of our approach.

Cultural distance is an important concept to study cultural differences, which is widely used in cross-culture study and cross-culture practice, such as enterprise internationalization [19]. Culture directly impacts on people's beliefs, thoughts, and social behaviors. While some people may not know this impact, culture and values have been rooted in people's minds and are the decisive factor in people's social behavior and their preferences. All computations of cultural distance are based on cultural dimensions.

The most famous cultural dimension research is Hofstede's cultural dimension theory, which first realizes the quantitative description of the abstract complicated concept of culture [20]. This cultural dimension theory enables researchers to compute cultures as data in a more intuitive way to compare differences between different cultures and various behaviors.

The Hofstede's cultural dimension theory consists of six dimensions. The six dimensions are as follows: Power Distance Index, Individualism versus Collectivism, Masculinity versus Femininity, Uncertainty Avoidance Index, Long Term Orientation versus Short Term Normative Orientation, and Indulgence versus Restraint. Based on Hofstede's cultural dimension theory, [21] first uses a simple mathematical formula to define the cultural distance and describes the cultural differences more simply. The formula is as follows:

\[
\text{CD}_j = \frac{m}{\sum_{i=1}^{m} \left( \frac{(I_{ij} - I_{ik})^2}{V_i} \right)},
\]

where \(\text{CD}_j\) is the cultural distance for the \(j\)th country, \(I_{ij}\) is Hofstede's \(i\)th cultural dimension score of the \(j\)th country, \(I_{ik}\) is the host country, \(V_i\) is the variance of the \(i\)th cultural dimension scores, and \(m\) is the number of cultural dimensions.

2.2. Related Work. A large number of service recommendation approaches have been proposed in the literature, and we only review some notable approaches here.

Among the numerous service recommendation approaches, collaborative filtering is one of the most successful recommendation technologies. And it includes user-based collaborative filtering [14, 22, 23], service-based collaborative filtering [24–28], and its fusion [18]. User-based collaborative filtering is to predict the rating of users based on the ratings of their similar users, and service-based collaborative filtering is to predict the rating of users based on the ratings of services which are similar to the services chosen by the users. The basic steps of collaborative filtering are rating collection, similarity computation, neighbor selection, rating prediction, and recommendation, and the similarity computation is always based on Person Correlation Coefficient [29]. Besides, [18] proposes a fusion approach which combines user-based and service-based collaborative filtering approaches. Different
from [18], our approach considers the cultural factor in rating prediction, and the similarity is not the only weigh factor of neighbors.

However, there are some problems in collaborative filtering approaches, such as the sparse rating data, cold start, malicious attack, and bad system scalability. To solve these problems, many trust-based recommendation approaches are proposed [16,17,30,31]. The paper [30] proposes a TPCF model which joins trust propagation into the collaborative recommendation system and combines the trust in trust propagation model and the similarity in traditional collaborative filtering algorithm by a mixing index to obtain the final user similarity. The paper [17] put forward a random walk algorithm which combines collaborative recommendation based on the item and the recommendation algorithm based on trust to predict the rating of a single item, and the principle is that the ratings of similar projects by highly trusted neighbors are more reliable than the ratings of the target project by lowly trusted neighbors. Besides, some recommendation approach is based on missing rating prediction [18] to improve the sparsity of rating data.

3. Our Approach

Aiming at the more reliable recommendation, we propose a cultural distance-aware service recommendation approach based on missing ratings prediction. Actually, the missing ratings prediction of training data is the same as the ratings prediction of active user in service collaborative filtering recommendation; therefore, we use the same cultural distance-aware collaborative filtering approach to solve them. The first step of our approach is the missing ratings prediction of training data which contains the similarity computation based on initial ratings, neighbor selection, and missing ratings prediction based on cultural distance; the second step is service recommendation which contains the similarity computation based on prediction rating data, neighbor selection, and ratings prediction based on cultural distance and recommendation.

3.1. Ratings Prediction. The foundation of the collaborative filtering recommendation approach is the user-service ratings matrix; however, these matrices are always sparse in practical applications. In our approach, we predict the missing rating values before recommendation to obtain a denser user-service ratings matrix. The main idea of prediction is similar to the collaborative filtering, and it can be divided into three phases: similarity computation, neighbor selection, and ratings prediction.

3.1.1. Similarity Computation. Assume that the original user-service ratings matrix $R$ which expresses users preferences contains $n$ users and $m$ services; the element in this two-dimensional matrix $r_{ui}$ is a vector of QoS values which is obtained by the user $u$ on the service $i$. In general, a user only invokes a small part of the services and rates less services; as a result, many elements $r_{ui}$ in $R$ are missing.

To predict the missing ratings, we adopt two collaborative filtering prediction approaches: the user-based prediction approach and the service-based prediction approach. The former is to first match the user’s other ratings against other users, compute the similarity between them, and find users with the most similar preferences. The latter is to first match the service’s other ratings against other services, compute the similarity between them, and find the most similar services. No matter in user-based or service-based prediction approaches, the similarity computation is based on Pearson correlation coefficient.

The Pearson correlation coefficient is a measure of the linear correlation between two variables. In user-based prediction approach, the similarities between user $u$ and other users can be represented as follows:

$$
\text{Sim}_u = \{ \text{sim}_{au} \mid a \in C, a \neq u \},
$$

where $C$ represents the set of users, $\text{sim}_{au}$ represents the similarity between user $a$ and user $u$, and it is computed as follows:

$$
\text{sim}_{au} = \frac{\sum_{i \in I} (r_{ai} - \overline{r}_a)(r_{ai} - \overline{r}_u)}{\sqrt{\sum_{i \in I} (r_{ai} - \overline{r}_a)^2} \sqrt{\sum_{i \in I} (r_{ai} - \overline{r}_u)^2}},
$$

where $I$ represents the subset of services which are rated by both user $a$ and user $u$, $\overline{r}_a$ represents the average of ratings obtained by the user $a$ on services in $I$, and $\overline{r}_u$ represents the average of ratings obtained by the user $u$ on the services in $I$. The value of $\text{sim}_{au}$ is between $-1$ and 1, where $-1$ represents total negative linear correlation, 0 represents no correlation, and 1 represents total positive linear correlation. The larger value of $\text{sim}_{au}$ indicates the higher similarity between user $a$ and user $u$. Note that when $I$ is an empty set, $\text{sim}_{au}$ is $-1$.

In service-based prediction approach, the similarities between service $i$ and other services can be represented as follows:

$$
\text{Sim}_i = \{ \text{sim}_{ij} \mid j \in S, j \neq i \},
$$

where $S$ represents the set of services, $\text{sim}_{ij}$ represents the similarity between service $i$ and service $j$, and it is computed as follows:

$$
\text{sim}_{ij} = \frac{\sum_{u \in U} (r_{ui} - \overline{r}_{U}) (r_{uj} - \overline{r}_{U})}{\sqrt{\sum_{u \in U} (r_{ui} - \overline{r}_{U})^2} \sqrt{\sum_{u \in U} (r_{uj} - \overline{r}_{U})^2}},
$$

where $U$ is the subset of users who rate both service $i$ and service $j$, $\overline{r}_{U}$ represents the average of ratings of service $i$ obtained by the users in $U$, $\overline{r}_{U}$ represents the average of ratings of service $j$ obtained by the users in $U$. Similar to $\text{sim}_{au}$, the value of $\text{sim}_{ij}$ is between $-1$ and 1, and the larger value of $\text{sim}_{ij}$ indicates the higher similarity between service $i$ and service $j$. Note that when $U$ is an empty set, $\text{sim}_{ij}$ is $-1$.

3.1.2. Neighbor Selection. The main idea of collaborative filtering is to harness the collective intelligence and learn from the opinions of correlative populations; therefore, it is important to select the neighbors which are similar users or similar services. Before ratings prediction, neighbor selection
of Top-$K$ it is not very similar to Point 1. Different from the principle Figure 1(a), Point 5 is selected as a neighbor of Point 1, but there are not enough similar points nearby. For example, in some points which are not very similar as neighbors when because the number of neighbors is fixed, and we have to take some points which are not very similar as neighbors when there are not enough similar points nearby. For example, in Figure 1(a), Point 5 is selected as a neighbor of Point 1, but it is not very similar to Point 1. Different from the principle of Top-$K$ approach, threshold-based approach is to select the users or services of which similarity is smaller than threshold as the neighbors. As shown in Figure 1(b) where $ts$ represents the value of threshold, to select the neighbors of Point 1 based on threshold, we consider the circular region with center Point 1, and a radius of $ts$, and take the points within the circular region as neighbors. In this example, Points 2, 3, 4, and 7 are selected as neighbors. This approach can improve the performance on the calculation of isolated points; however, this approach is very sensitive to the value of threshold; the neighbors selected are few when $ts$ is too small, and when $ts$ is too large. Therefore, how to set the value of threshold is very difficult.

To ensure the similarity of neighbors, we propose a novel neighbor selection approach which combines the Top-$K$ approach and the threshold-based approach. We first set a loose threshold to filter the users or services with low similarity and then select the remaining users or services which are the top $K$ most similar users or services as neighbors.

In user-based prediction approach, the selected neighbors of user $u$ can be described as follows:

$$N_u = \{ a | \text{sim}_{au} < ts, \ r_{au} < K \},$$

(6)

where $N_u$ represents the neighbors of user $u$ and $r_{au}$ represents the ranking of $\text{sim}_{au}$ among $\text{Sim}_u$. The formulation means that user $a$ is a neighbor of user $u$ if user $a$ belongs to the top $K$ similar users of user $u$, and $\text{sim}_{au}$ is less than the threshold.

In service-based prediction approach, the selected neighbors can be described as follows:

$$N_i = \{ j | \text{sim}_{ij} < ts, \ r_{ij} < K \},$$

(7)

where $N_i$ represents the neighbors of service $i$ and $r_{ij}$ represents the ranking of $\text{sim}_{ij}$ among $\text{Sim}_i$. The formulation means that service $j$ is a neighbor of service $i$ if service $j$ belongs to the top $K$ similar users of service $i$, and $\text{sim}_{ij}$ is less than the threshold.

3.1.3. Ratings Prediction. In traditional collaborative filtering approaches, the similarity between users or services is the only factor to determine the accuracy of prediction results. However, there are other factors having an important role in prediction. Therefore, we adopt the cultural distance in our collaborative filtering approach to improve the accuracy of prediction.

In this paper, we design a cultural factor to represent the preference of users to their neighbors. The cultural factor between user $u$ and its neighbor $a$ is as follows:

$$c_{au} = \left(1 - \frac{|cd_{au} - cd_{a}|}{\max_{cd_{eu}} - \min_{cd_{eu}}} \right),$$

(8)

where $c_{au}$ represents the cultural factor of user $u$ to user $a$, $cd_{au}$ represents the cultural distance value of user $u$, $\max_{cd_{eu}}$ represents the maximum value among the cultural distance values of neighbors of user $u$, and $\min_{cd_{eu}}$ represents the minimum value among the cultural distance values of neighbors of user $u$.

Similarly, the cultural factor between service $i$ and its neighbor $j$ is as follows:

$$c_{ij} = \left(1 - \frac{|cd_{i} - cd_{j}|}{\max_{cd_{di}} - \min_{cd_{di}}} \right),$$

(9)

where $c_{ij}$ represents the cultural factor of service $i$ to service $j$, $cd_{i}$ represents the cultural distance value of service $i$, $\max_{cd_{di}}$ represents the cultural distance value of service $j$, $\max_{cd_{di}}$
represents the maximum value among the cultural distance values of neighbors of service $i$, $\min_{a,b} \theta_i$ represents the minimum value among the cultural distance values of neighbors of service $i$.

In our missing ratings prediction approach, we design a hybrid weight which can consider not only the similarity of neighbors but also the cultural factor. In our user-based prediction approach, the compound weight between user $u$ and its neighbor $a$ is

$$w_{au} = \beta \sin_{au} + (1 - \beta) c_{au},$$

(10)

where $\beta$ represents the degree of the emphasis on the similarity. And then the prediction value of missing ratings can be computed based on the ratings of neighbors as follows:

$$p_u (r_{ui}) = \bar{r}_u + \frac{\sum_{a \in N_u} w_{au} (r_{ai} - \bar{r}_u)}{\sum_{a \in N_u} w_{au}},$$

(11)

where $p_u (r_{ui})$ represents the prediction value of the missing rating $r_{ui}$ based on the neighbor users, $\bar{r}_u$ represents the basic prediction value of $r_{ui}$, and it is the average of ratings obtained by the user $u$; similarly, $\bar{r}_i$ is the average of ratings obtained by the user $a$ on the services besides service $i$.

Similarly, in our service-based prediction approach, the compound weight between service $i$ and its neighbor $j$ is

$$w_{ij} = \beta \sin_{ij} + (1 - \beta) c_{ij},$$

(12)

And then the prediction value of missing ratings can be computed based on the ratings of similar services as follows:

$$p_s (r_{ai}) = \bar{r}_j + \frac{\sum_{j \in N_i} w_{ij} (r_{aj} - \bar{r}_j)}{\sum_{j \in N_i} w_{ij}},$$

(13)

where $p_s (r_{ai})$ represents the prediction value of the missing rating $r_{ai}$ based on the similar neighbor services, $\bar{r}_i$ represents the average of ratings of service $i$, and $\bar{r}_j$ represents the average of ratings of service $j$ besides the rating obtained by user $u$.

In order to take full advantage of the information of the similar users and similar services and ensure the diversity of the recommendation results, our prediction approach combines the two collaborative filtering prediction approaches. Therefore, the prediction value of the missing rating $r_{ai}$ in our prediction approach is as follows:

$$p (r_{ai}) = \lambda p_u (r_{ai}) + (1 - \lambda) p_s (r_{ai}),$$

(14)

where $\lambda$ represents the weight of the user-based prediction approach.

3.2. Service Recommendation. After the missing ratings prediction, we can recommend services for the active users who require service recommendations based on the prediction rating data of the training users whose ratings have been obtained. Similar to our missing ratings prediction approach, our service recommendation approach can be divided into three phases: similarity computation, neighbor selection, ratings prediction, and recommendation.

The first phase is similar to the last section. The similarities between the active user and the training users and the similarities between the candidate services and other services are computed by Formula (3) and Formula (5). The second phase is to select neighbors for the active users and the candidate services by Formula (6) and Formula (7). The third phase is to predict the ratings of the candidate services from the active users by Formula (14), and the final phase is to recommend service according to the prediction ratings.

4. Experiment

In this section, we implement our cultural distance-based collaborative filtering recommendation approach (named CDCF) to verify the performance and comparatively evaluate CDCF against several service recommendation approaches in terms of the prediction failure rate under the training rating data with different density and the active rating data with different density. The experimental evaluation results show that CDCF can improve the accuracy of recommendation.

4.1. Experiment Setup. To evaluate the performance of CDCF, we implement it by adopting two real-world datasets: WS-Dream dataset [32] and Hofstede’s cultural dimension dataset [33].

The WS-Dream dataset contains the real-world QoS evaluation results from 339 users on 5,825 Web services and the location information (e.g., IP address, country, region, latitude, and longitude) of these users and services. Specifically, the QoS evaluation results are the user-service matrices on response time and throughout which can be used as the rating matrix in our approach.

Hofstede’s cultural dimension dataset has been published in [33] and it contains the scores for the six cultural dimensions. These scores are obtained according to the items in the IBM database plus extensions and the items in the World Values Survey.

Based on the two datasets, we compare CDCF with other recommendation approaches.

(i) User-Based Collaborative Filtering Recommendation Approach (UCF). This approach is to predict the missing ratings using the traditional user-based prediction approach without adopting the cultural distance firstly and then recommend services for active users by employing the traditional user-based collaborative filtering approach based on the prediction ratings data.

(ii) Service-Based Collaborative Filtering Recommendation Approach (SCF). This approach is to predict the missing ratings using the prediction approach without adopting the cultural distance firstly and then recommend services for active users by employing the traditional service-based collaborative filtering approach based on the prediction ratings data.

(iii) Traditional Hybrid Collaborative Filtering Recommendation Approach (TCF). This approach is to predict the missing ratings by combining the traditional user-based
prediction approach and traditional service-based prediction approach and then recommend services for active users by combining the traditional user-based approach and service-based prediction approach. Compared with CDCF, both of the prediction and recommendation approaches only depend on the similarity of neighbors.

We select 150 users from the WS-Dream dataset randomly and then divide them into training users and the active users; the rate of training users is 0.75 in our experiments. To evaluate the performance of CDCF in data with different density, we randomly remove some ratings in training rating matrix or active rating matrix and employ different approaches to verify the reliability of CDCF.

The evaluation index used in this paper is Root Mean Square Error (RMSE) to measure the recommendation reliability of CDCF in comparison with other collaborative filtering approaches. RMSE is defined as follows:

\[
RMSE = \sqrt{\frac{\sum (r_{ui} - \bar{r}_{ui})^2}{N}},
\]

where \( r_{ui} \) represents the removed rating of active user \( u \), \( \bar{r}_{ui} \) represents the prediction value, and \( N \) represents the number of the removed ratings.

4.2. Performance Comparison. Figure 2 shows the comparison results in terms of RMSE with different density of training data ranging from 0.1 to 0.3, while the density of active data is 0.3. As shown in Figure 2, our approach CDCF is much better than other approaches, and its RMSE is 22.2%, 7.5%, and 13.4% lower than SCF, UCF, and TCF on average. In addition, there is no obvious relationship between the recommendation reliability and the density of training data in SCF, UCF, TCF, and CDCF.

In a word, as shown in Figures 2 and 3, although TCF takes full advantage of the information of the similar users and similar services and ensures the diversity of the recommendation results, its accuracy is lower than SCF with the low accuracy of UCF. Our approach CDCF can ensure not only the diversity of the recommendation results, but also the high recommendation reliability.

5. Conclusion

In this paper, we propose a cultural distance-aware service recommendation approach based on missing rating prediction. The main idea is to adopt the cultural distance to represent the user preference and combine the cultural factor with similarity to improve the reliability of recommendation. In addition, we predict the missing ratings by cultural distance-aware collaborative filtering approach to improve the sparsity of rating data. Experimental results show that compared to previous approaches, our approach can significantly improve the accuracy of the recommendation results. However, the cultural factor is at the regional/national level in this paper. A narrower grouping is required to improve the reliability of recommendation. Consequently, the next study in this area could focus on the presentation of the user cultural factor in a narrower group.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The work presented in this study was supported by the National Natural Science Foundation of China (Grant no.
References


Nonmetric Correction of Lens Distortion Based on Entropy Measure

Tianfei Chen, Lijun Sun, Qiuwen Zhang, Xiang Wu, and Defeng Wu

1School of Electrical Engineering, Henan University of Technology, Zhengzhou, Henan 450001, China
2College of Computer and Communication Engineering, Zhengzhou University of Light Industry, Zhengzhou, Henan 450002, China
3Marine Engineering Institute, Jimei University, Xiamen, Fujian 361021, China

Correspondence should be addressed to Lijun Sun; sunlijunzz@163.com

Received 17 October 2017; Accepted 3 December 2017; Published 4 February 2018

In the real vision system, lens always inevitably contains nonlinear distortion, which leads to geometric distortion of digital image, so it must be corrected. In this paper, a nonmetric correction algorithm for lens distortion based on entropy measure is proposed. The algorithm uses the imaging characteristics of the space line in the ideal perspective model, and the distortion entropy is defined to measure the degree of lens distortion. For distortion curves with different distribution, the calculation dimension of distortion entropy measure is uniform, which can reduce the influence of curve inhomogeneity. On this basis, the modified distortion entropy measure with normalized weight is put forward to enhance the capability of noise suppression, and the distortion correction performance of the traditional interior point optimization algorithm, basic artificial bee colony (ABC) algorithm, and Gbest-guided artificial bee colony (GABC) algorithm is compared and analyzed. The simulation experiments demonstrate that the correction performance of GABC to optimize the modified distortion entropy measure with normalized weight is best, and it has strong robustness to noise. Finally, the actual image distortion correction examples verify the effectiveness of the proposed algorithm.

1. Introduction

In the computer vision, vision inspection, image processing, and other research fields, ideal pinhole model is often used to represent camera lens. However, the manufacturing or installation factors in the actual production conditions lead to the phenomenon that the captured digital image has geometric distortion; thus the pinhole camera model is far from enough especially for some low-cost or wide-angle lenses [1]. In order to ensure the accuracy of subsequent vision processing and analysis, the actual distortion of camera lens should be described and corrected.

At present, the algorithm for lens distortion correction can be mainly divided into metric method and nonmetric method. The metric methods [2, 3] require the accurate location of feature points in space, and the distortion model parameters are also integrated into the camera model. These methods establish the corresponding relationship between feature points in world coordinate system and the two-dimensional coordinates in CCD image, and the distortion model parameters and camera model parameters are together obtained by nonlinear optimization. Therefore, these methods have more parameters and require large computation and some known coordinates of feature points; thus the correction cost is higher. In addition, the coupling of distortion model parameters and camera model parameters can easily occur in the nonlinear optimization process, which can affect the distortion correction performance.

The nonmetric correction methods [4] make full use of the geometric invariants of ideal pinhole camera model. Generally, these geometric invariants represent the properties of structures and remain unchanged under appropriate transformations; thus these methods have no need to require some known structure reference. Currently, the available features of geometric invariants include straight line [5, 6], cross ratio [7, 8], vanishing points [9], and plane constraints [10]. The straight line is the most widely used geometric feature, and the projection of straight line should be straight under any perspective transformation. Therefore, [11] uses col-linear feature points to measure distortion, and col-linear vector is...
adopted in [12]. In addition, geometric feature of straight line also contains line slope [13] and three mutually orthogonal sets of parallel lines [14]. The distortion correction algorithm based on cross ratio invariability is proposed by He et al. [8], which only requires image coordinates of four colinear points and cross ratio. However, it has some limitations, because only one-order distortion model is considered. On the basis of cross ratio invariability, the linear equations are introduced by Ricolfe-Viala and Sánchez-Salmerón [15], which make the correction method model more robust. Zhao et al. [16] have analyzed the accuracy of correction method based cross ratio invariability and propose the distortion correction algorithm using colinear constraint of vanishing points. Wang et al. [17] use the division distortion model with single parameters, and the distorted image points are adopted for circle fitting; thus this method is limited by the circle fitting accuracy. Reference [18] uses multiview corresponding points to realize distortion correction, but mismatching points can lead to poor correction performance. When the distortion measure has been defined using geometric invariants, the distortion parameters can be finally solved by nonlinear optimization algorithm. The traditional nonlinear optimization algorithm has strong dependence on the initial value, and the setting of initial value has a direct impact on the optimization accuracy, such as Levenberg-Marquardt optimization method and Interior Point Method with constrains. As a research issue in the field of artificial intelligence, the swarm intelligence algorithm has become a new method to solve the traditional optimization problems in recent years. It does not need to set initial value of iteration, and the whole intelligence behavior is carried on through the cooperation of simple individuals, such as genetic algorithm, particle swarm optimization, and artificial bee colony algorithm. Zhang et al. [19] and Chen et al. [20] have in respective successfully applied genetic algorithm and particle swarm optimization algorithm in lens distortion correction. As a novel swarm intelligence algorithm, the artificial bee colony (ABC) algorithm has simple structure, less control parameters, and other characteristic, and it also has better performance in numerical optimization and other application fields [21].

In this paper, the concept of entropy is introduced to measure the degree of lens distortion by using the geometric characteristics of straight line in pinhole camera model. The new definition of distortion entropy has uniform calculation dimension, which can reduce the influence of uneven distribution of distortion curve. Besides that, the distortion entropy measure is further improved to enhance the ability of noise suppression in the analysis of curvature characteristic of distortion curve. The algorithm of ABCs is adopted to carry on the final nonlinear optimization, and the correction performance is compared and analyzed among the traditional interior point optimization, basic ABC, and GABC. In the proposed algorithm, the defined distortion entropy measure is simple and easy to implement, and the modified entropy measure with normalized weight has the noise suppression ability to some extent. In addition, the algorithm of GABC not only has no need to set initial value of iteration but also has better optimization accuracy and faster convergence speed. Finally, a large number of simulation and actual experiments verify the correction performance of the proposed algorithm.

2. Lens Distortion Model and Distortion Measure

The actual camera lens generally has different degrees of distortion due to manufacturing, assembly, and other factors, which can lead to the phenomenon that the actual image points are not identical to the ideal image points, and the difference between them is called lens distortion. Usually, the lens distortion is classified into radial distortion and tangential distortion, and the radial distortion is dominant [2].

2.1. Lens Distortion Model. In general, lens distortion can be quantitatively described using a mathematical model, and the lens distortion model also can be considered as mapping function between actual image points \((x^d, y^d)\) and ideal image points \((x^u, y^u)\), where

\[
F : (x^d, y^d) \rightarrow (x^u, y^u).
\]

After analyzing the actual imaging process, Brown [22] presented the mathematical expression of radial and tangential distortion model, and it has become the traditional lens distortion model; the formulation is expressed as follows:

\[
\begin{align*}
\frac{x^u}{x} &= x^d + x^d \left( k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots \right) \\
&\quad + \left[ \rho_1 \left( r^2 + 2x^2 \right) + 2 \rho_2 x^2 y^2 \right] \left[ 1 + p_3 r^2 + \cdots \right] \\
\frac{y^u}{y} &= y^d + y^d \left( k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots \right) \\
&\quad + \left[ \rho_2 \left( r^2 + 2y^2 \right) + 2 \rho_1 x^2 y^2 \right] \left[ 1 + p_3 r^2 + \cdots \right]
\end{align*}
\]

(2)

where \(x^d = x - u_0, y^d = y - v_0, r^2 = x^2 + y^2, (u_0, v_0)\) is the distortion center. \(k_1, k_2, \ldots\) are radial distortion parameters, and \(p_1, p_2, \ldots\) are tangential distortion parameters.

However, in practical applications, less parameters cannot fully express the complexity of lens distortion, and too many parameters will also cause the instability of the solution [4]. Therefore, two radial distortion parameters and two tangential parameters are most adopted which can satisfy most lenses with different distortion degree. The model is expressed as

\[
\begin{align*}
\frac{x^u}{x} &= x^d + x^d \left( k_1 r^2 + k_2 r^4 \right) \\
&\quad + \left[ \rho_1 \left( r^2 + 2x^2 \right) + 2 \rho_2 x^2 y^2 \right] \\
\frac{y^u}{y} &= y^d + y^d \left( k_1 r^2 + k_2 r^4 \right) \\
&\quad + \left[ \rho_2 \left( r^2 + 2y^2 \right) + 2 \rho_1 x^2 y^2 \right].
\end{align*}
\]

(3)

If all the distortion parameters are known, lens distortion can be corrected through the above distortion model.
2.2. Distribution Law of Distortion Curve. If camera lens have no distortion, the imaging projection of straight lines in three-dimensional (3D) space should be straight based on the pinhole camera model and perspective projection principle. In fact, the actual projection of straight lines in space will deform and become distortion curves due to the inevitable distortion. In order to analyze the change law of distortion curves with different distribution influenced by lens distortion, some ideal horizontal lines are generated at intermediate intervals in the CCD image, and equal interval sampling is performed on each line. If the distortion parameters are known, the corresponding distortion points can be calculated by using formula (3). Therefore, the distortion model parameters are solved by using a nonlinear optimization algorithm to minimize the defined distortion measure. When the correction effect is closer to the ideal straight line, the distortion parameters is closer to the true value. These methods do not need to have accurately manufactured structure reference and belong to nonmetric correction method.

At present, the traditional nonmetric correction methods usually define the sum of square for the distance from the corrected points to the corresponding fitting line as distortion measure. This distortion measure not only requires the fitting of multiple lines but also needs the distance calculation from each corrected point to fitting lines; hence, the computation is relatively large. Besides that, the distribution of distortion curve in CCD and the number of sampling points will also influence the sum of distance square, which make the calculation of distortion measure for each distortion curve nonuniform, then the actual correction effect can be influenced.

Entropy is a very active physical concept in recent years and has been widely used in many fields of natural and social sciences. In this paper, the distortion entropy measure is defined to describe the nonuniformity and bending degree of distortion curve. Because the concept of generalized entropy can usually be defined by the ratio between extensive quantity and intensity quantity, then the intensity quantity of the distortion entropy measure is represented by the distance between the first and the last points of distortion curves, and the extensive quantity of the distortion entropy measure is expressed by the sum of distances between all adjacent sampling points on the distortion curve. The specific formula is as follows:

\[
\xi_1(k_1, k_2, p_1, p_2, u_0, v_0) = \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{N} \left| p_{ij}^u - p_{ij+1}^u \right|, \tag{4}
\]

where \( M, N \) are in respective the number of distortion curves and the number of sampling points on each curve. \( p_{ij}^u \) represents the \( i_{th} \) corrected point on the \( l_{th} \) distortion curve, and \( p_{ij}^u, p_{ij}^v \) are, respectively, the first and the last corrected point on the \( l_{th} \) distortion curve. As can be seen from formula (4), the calculation dimension of distortion entropy for each distortion curve is uniform, and its value is close to 1, when the camera trends to be ideal pinhole model.

If the coordinates of the actual imaging points \( p_{ij}^u \) contain extraction noise, the distortion model (3) will amplify the extraction noise which may influence the coordinates of corrected points \( p_{ij}^u \), and the amplification is more obvious when the point far away from the distortion center. In order
to reduce the influence of extraction noise, the distortion entropy measure (4) is further improved, which is given by

$$\xi_k(k_1, k_2, p_1, p_2, u_0, v_0) = \sum_{i=1}^{M} w_i \cdot \frac{\sum_{j=1}^{N} \| p_{ij} - p_{ij,\text{start}} \|}{\| p_{ij,\text{start}} - p_{ij,\text{end}} \|}, \quad (5)$$

where $w_i$ represents the normalized weight for each distortion curve, which is related to the curvature of the distortion curve, and it can be expressed as

$$w_i = \frac{1/\xi_i}{\sum_{j=1}^{M} (1/\xi_j)}. \quad (6)$$

In the formula, $\xi_i$ denotes the mean curvature of the $i_{th}$ distortion curve.

### 3. Optimization Method

**3.1. Algorithm Description.** Artificial bee colony (ABC) algorithm is a kind of swarm intelligence algorithm proposed by Karaboga in 2005 [23, 24], and it has received extensive attention and research from many scholars due to its simple structure, less parameters, and easy implementation. Similar to other evolutionary algorithms, ABC algorithm carries out the iteration search based on a random generated swarm population, and the swarm includes three different kinds of bees, they are, respectively, employed bees, onlooker bees, and scout bees; then the information can be exchanged according to their respective division of labor. Assume that the number of food sources is $SN$, one of food sources $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D})$ represents a candidate solution, $i \in \{1,2,\ldots,SN\}$, and $D$ is the dimension of the optimization problem. The food source corresponds to the employed bees, and the number of employed bees is equal to that of onlooker bees. Then, the search process of ABC for the best food source can be described as follows:

1. The employed bees search in the neighborhood of current food sources and generate a new food source; then a better food source should be chosen greedily.
2. Food sources are selected by the onlooker bees according to the information provided by the employed bees, and they search for a new food source in the neighborhood; then current food source should be greedily replaced with a better one.
3. If the food source has not been updated continuously for $limit$ times, the current food source should be abandoned, and the employed bee becomes a scout bee and randomly searches for a new food source.

The above three phases should be repeated until the number of maximum iterations $MCN$ or the value of allowable error $\varepsilon$ are reached.

In the iterative searching process, the onlooker bees use the information provided by the employed bees to select food source by means of roulette wheel, and the selected probability of each food source is calculated according to the following formula:

$$P_i = \frac{fit_i}{\sum_{i=1}^{SN} fit_i}, \quad (7)$$

where $fit_i$ is the fitness value of the $i_{th}$ solution, and the fitness value is used to determine the quality of food source. That is to say, the larger the fitness value is, the better the food source is.

In addition, the fitness function is determined corresponding to the optimization problem. If the optimization problem is to find minimum value, the fitness function $fit(\cdot)$ is the deformation of the objective function $f(\cdot)$, which is usually expressed as follows:

$$fit(X_i) = \begin{cases} 1 & f(X_i) \geq 0 \\ \frac{1}{1 + |f(X_i)|} & f(X_i) < 0 \end{cases} \quad (8)$$

On the contrary, the fitness function is the objective function when the optimization problem is to find maximum problem.

The ABC algorithm performs the greedy selection according to following formula:

$$V_i = \begin{cases} V_i & fit(V_i) > fit(X_i) \\ X_i & fit(V_i) \leq fit(X_i) \end{cases} \quad (9)$$

where $V_i$ is the new food source generated in the neighborhood of the current food source.

In the ABC algorithm, the employed bees and the onlooker bees search for new food source according the following formula:

$$v_{i,j} = x_{i,j} + \phi_{i,j}(x_{i,j} - x_{k,j}), \quad (10)$$

where $\phi_{i,j}$ is a uniform random number in $[-1,1]$. $k \in \{1,2,\ldots,SN\}$ is randomly chosen indices, and $k \neq i$, $j \in \{1,2,\ldots,D\}$ indicates a random selection dimension.

When the fitness value corresponding to the employed bee cannot be improved continuously for $limit$ times, the employed bee will become the scout bee, then the scout bee produces a new food source according the following expression:

$$x_{i,j} = x_{\text{min},j} + \text{rand}(0,1) \cdot (x_{\text{max},j} - x_{\text{min},j}), \quad (11)$$

In the above expression, $x_{\text{max},j}$ and $x_{\text{min},j}$ are the upper and lower bounds for dimension $j$, respectively.

In the basic ABC algorithm, the bee swarm depends on the shared information of individuals to search for new food sources, and its randomness is greater, thus it is generally believed that the ABC algorithm has strong global search capability. In order to enhance the local search ability of ABC algorithm, the global best- (gbest-) guided ABC is proposed by Zhu and Kwong (GABC) [25], and a search operator guided by the best solution is introduced to improve the exploitation ability of the algorithm. Thus, the corresponding search equation is expressed as follows:

$$v_{i,j} = x_{i,j} + \phi_{i,j}(x_{i,j} - x_{k,j}) + \phi_{i,j}(\text{gbest}_j - x_{i,j}), \quad (12)$$
where parameter $\psi$ is uniformly random number distributed in $[0, C]$ and $C$ is nonnegative constant. Generally, it is set to 2. The remaining parameters are the same as those in formula (10).

3.2. Algorithm Framework. After the description of ABC\ GABC algorithm, the framework of the corresponding algorithm is given below. The ABC algorithm uses formula (10) to generate new food source, and GABC uses formula (12), then the rest of algorithm flow is the same. \( FEs \) indicates the current evaluation times of fitness function, and \( MaxFEs \) is the maximum number of fitness function evaluations. \( tria1 \) is used to record the number of times of food source \( X_i \) which has not been updated.

\textit{ABC\ GABC Framework}

1. Generate \( SN \) food sources \( \{X_i \mid i = 1, 2, \ldots, SN\} \) randomly
2. \( FEs = SN \);
3. While \( FEs \leq MaxFEs \) do
4. % Employed Bee Phase
5. for \( i = 1 \) to \( SN \) do
6. Generate a new candidate solution \( V_i \) according to Eq. (10) or Eq. (12);
7. Update candidate solution \( V_i \) according to Eq. (9);
8. if \( f(V_i) < f(X_i) \), set \( tria1_i = 0 \), else \( tria1_i = tria1_i + 1 \); end
9. \( FEs = FEs + 1 \);
10. end
11. % Onlooker Bee Phase
12. for \( ii = 1 \) to \( SN \) do
13. Choose a food source \( X_i \mid i \in 1, 2, \ldots, SN \) from the current swarm by the roulette wheel selection mechanism;
14. Generate a new candidate solution \( V_i \) according to Eq. (10) or Eq. (12);
15. Update candidate solution \( V_i \) according to Eq. (9);
16. if set \( tria1_i = 0 \), else \( tria1_i = tria1_i + 1 \); end
17. \( FEs = FEs + 1 \);
18. end
19. % Scout Bee Phase
20. if \( tria1_i > \text{limit} \), replace \( X_i \) with a newly randomly candidate.
21. if \( FEs > MaxFEs \), stop and output the best solution achieved so far, otherwise, go to step (3)
22. Output the best solution.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image.png}
\caption{Randomly generated synthetic data.}
\end{figure}

4. Experiments and Results

In this section, the performance of the technique is assessed using both synthetic and real image data. Since the synthetic image provides exact knowledge of line positions, distortion center, and distortion parameters, the precise quantitative evaluation of performance is possible. The performance on real images is shown to demonstrate the practical implementation of the technique.

4.1. Tests on Synthetic Data. Assume that the resolution of the simulation image is set as \( 768 \times 576 \), the distortion center \((u_0, v_0)\) is \((380, 280)\) and the distortion model parameters are, respectively, \( k_1 = 6 \times 10^{-6}, k_2 = 0, p_1 = -2 \times 10^{-7}, p_2 = 0 \). Ten simulation ideal lines are randomly generated in the image, and the sampling points are evenly removed on each line, then the corresponding distortion points can be computed based on the distortion model (3). Figure 3 shows the randomly generated synthetic data, and the circle represents the distortion center, the solid points are the sampling point on the ideal line, and the cross point is the corresponding distortion point on actual curve. When the distortion measure function has been defined, the work of lens distortion correction can be transformed into a nonlinear optimization problem which regards the distortion measure as the objective function. In order to reduce the unnecessary search range for nonlinear optimization algorithms, the solution space of the distortion model parameters can be limited as follows: \( k_1 \in [-10^{-5}, 10^{-5}], k_2 \in [-10^{-9}, 10^{-9}], p_1 \in [-10^{-5}, 10^{-5}], p_2 \in [-10^{-5}, 10^{-5}] \). The above limitation space can still satisfy the requirement of large distortion image. The simulation experiment is realized on the computer with main frequency 3.6 GHz, memory 8 G, Matlab R2014a, and Windows 7 operating system.

Firstly, the correction performance of the traditional nonlinear optimization algorithm and the ABC global optimization algorithm is compared and analyzed. The traditional nonlinear optimization uses the Matlab optimization toolbox function \texttt{fmincon}, which selects the Interior Point Method (IPM) to solve the problem. The initial value for the IPM is
set as follows: \( k_1 = k_2 = p_1 = p_2 = 0 \), and the initial value of distortion center is generally set as the geometric center of the image, that is, \([u_0, v_0] = [381, 288] \). However, the ABC algorithm is a nonlinear global optimization algorithm based on swarm intelligence, which do not need to set the initial value, and the control parameters of the ABC are set as follows: the size of swarm population is \( np = 50 \); the maximum number of function evaluation \( MaxFEs \) is 10000, so the maximum of iterations \( MaxIt \) is 200; because the dimension of correction problem \( ndim \) is 6, the control parameter \( limit \) is set as \( 0.1 \times np \times ndim \). As the distortion model is a highly nonlinear model, for example, coupling exists between distortion center and distortion parameters, different distortion parameters can cause similar correction results, so it is meaningless to compare the distortion parameters directly [4]. Therefore, the correction performance is evaluated using the root-mean-square (RMS) error between the coordinates of ideal points and those of the corrected points. In order to simulate the feature points extraction error, the Gauss white noise with standard deviation \( \sigma \) is added in the feature points, then the standard deviation \( \sigma \) is increased by 0.25, and 30 times running is carried on independently at each noise intensity. Finally, the average value of RMS can be obtained.

Figure 4 shows the correction result of IPM and ABC algorithm under the distortion measure \( \xi_1 \), it can be seen from that the correction error increases with the increase of noise intensity, and the correction effect of ABC is much better than that of IPM algorithm. The reason is that the IPM algorithm is a local search algorithm which does not have the global search ability, and it depends on the initial value setting, so it is easy to fall into local optimum value. Therefore, its correction effect is worse. However, the ABC algorithm has strong exploration ability through individual cooperation, and it has been proved theoretically that the ABC algorithm has global convergence [26].

In order to compare the correction performance of ABC and GABC under the distortion measures \( \xi_1 \) and \( \xi_2 \), 30 times running is carried on independently at each noise intensity, and then the average value of RMS is also computed. The comparative results are shown in Figure 5. The setting of core control parameters for GABC is consistent with ABC. Besides that, the control parameters \( C \) of GABC are set as 2 according to [25]. From the comparative results, the following two conclusions can be obtained. Conclusion 1: distortion measure \( \xi_2 \) has better noise suppression ability than the distortion measure \( \xi_1 \), because the distortion model will amplify the extraction noise of points and the amplification is more obvious when the point is far away from the distortion center. According to the different distribution of distortion curves, the normalized weights for distortion measure \( \xi_2 \) are determined to suppress the noise influence. Conclusion 2: the correction result of GABC is better than that of ABC. Although the ABC has strong exploration ability, its local search ability is insufficient, and it is easy to cause premature convergence. However, the current best solution is integrated into the search equation to improve the local search ability while maintaining the global search ability; thus it can be easier to obtain the global optimal solution for GABC. Figure 6 shows the evolutionary curves of mean value of distortion measure for GABC and ABC at 30 times independent running. It can be concluded that the convergence accuracy of GABC is better than that of ABC, and the GABC also has a faster convergence rate.

To sum up the above simulation results, the correction performance using distortion measure \( \xi_2 \) and GABC algorithm is much better. Besides that, it has strong suppression effect on noise and is of more robustness.

4.2. Tests on Real Images. According to the above conclusion from the simulation analysis, we use distortion measure \( \xi_2 \) and GABC algorithm to correct actual images. Figure 7 shows the captured checkerboard image. The coordinates of corner points can be extracted by OpenCV function, and its accuracy is 0.1 pixels, then the correction algorithm is
Table 1: Distortion model parameters.

<table>
<thead>
<tr>
<th>$k_1$</th>
<th>$k_2$</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$u_0$</th>
<th>$v_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0747e-06</td>
<td>1.6961e-12</td>
<td>3.7833e-06</td>
<td>3.2583e-06</td>
<td>398.6734</td>
<td>286.8387</td>
</tr>
</tbody>
</table>

Table 2: Analysis data of col-linear points.

<table>
<thead>
<tr>
<th>Line number</th>
<th>Max value</th>
<th>Min value</th>
<th>Mean value</th>
<th>Max value</th>
<th>Min value</th>
<th>Mean value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>7.9615</td>
<td>0.3657</td>
<td>4.1767</td>
<td>0.1401</td>
<td>0.0055</td>
<td>0.0592</td>
</tr>
<tr>
<td>(2)</td>
<td>6.0414</td>
<td>0.1576</td>
<td>3.1271</td>
<td>0.1100</td>
<td>0.0127</td>
<td>0.0585</td>
</tr>
<tr>
<td>(3)</td>
<td>3.8724</td>
<td>0.0987</td>
<td>1.9783</td>
<td>0.1200</td>
<td>0.0064</td>
<td>0.0454</td>
</tr>
<tr>
<td>(4)</td>
<td>1.3082</td>
<td>0.0314</td>
<td>0.6584</td>
<td>0.1184</td>
<td>0.0014</td>
<td>0.0470</td>
</tr>
<tr>
<td>(5)</td>
<td>1.4839</td>
<td>0.0223</td>
<td>0.7679</td>
<td>0.0525</td>
<td>7.019e-4</td>
<td>0.0203</td>
</tr>
<tr>
<td>(6)</td>
<td>4.0687</td>
<td>0.1421</td>
<td>2.1119</td>
<td>0.0375</td>
<td>0.0064</td>
<td>0.0209</td>
</tr>
<tr>
<td>(7)</td>
<td>6.3782</td>
<td>0.1817</td>
<td>3.3073</td>
<td>0.0797</td>
<td>4.8226e-4</td>
<td>0.0366</td>
</tr>
<tr>
<td>(8)</td>
<td>8.2823</td>
<td>0.3479</td>
<td>4.3206</td>
<td>0.0789</td>
<td>0.0026</td>
<td>0.0319</td>
</tr>
</tbody>
</table>

carried on to solve the parameters of the distortion model. Table 1 is the obtained parameters of the distortion model. The corrected image is shown in Figure 8, and Figure 9 shows the distortion points and corresponding corrected points, the solid points represent the distortion points, and the cross indicates the corrected points. Table 2 is the analysis data of col-linear points, the Max, Min, and Mean, respectively, represent the maximum distance, the minimum distance, and
the average distance from the points to the fitting line. From the comparative result of Figures 7–9, the correction effect is obvious. From the analysis data in Table 2, we can see that all of the distance data after correction is much better than before.

Figures 10 and 11 are, respectively, the scene images and corresponding corrected images. It can be seen that the edge of window and the cabinet are well corrected. In summary, the effectiveness of the proposed algorithm is demonstrated.

5. Conclusion

A nonmetric correction algorithm for lens distortion based on entropy measure is proposed in this paper. The distortion entropy measure is defined to describe the nonuniformity and bending degree of distortion curve, which can reduce the influence of curve inhomogeneity. On this basis, the modified distortion entropy measure with normalized weight is put forward to enhance the capability of noise suppression. Then, the distortion correction performance of the traditional interior point optimization algorithm, basic ABC algorithm, and GABC algorithm is compared and analyzed. The simulation experiments demonstrate that the correction performance of GABC to optimize the modified distortion entropy measure with normalized weight is best, and it has strong robustness to noise. Finally, the actual image distortion correction examples verify the effectiveness of the proposed algorithm.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (U1604151 and 61302118), Outstanding Talent Project of Science and Technology Innovation in Henan Province (174200510008), Program for Scientific and Technological Innovation Team in Universities of Henan Province (16IRTSTHN029), Program for Science and Technology Innovation Talents in Universities of Henan Province (17HASTIT022), the Funding Scheme of Young Key Teacher of Henan Province Universities (2016GGJS-087), and the Fundamental Research Funds of Henan University of Technology (2015QNJJH13 and 2016XTCX06).

References


Research Article

Field Geometric Calibration Method for Line Structured Light Sensor Using Single Circular Target

Tianfei Chen,1 Lijun Sun,1 Qiuwen Zhang,2 Xiang Wu,1 and Defeng Wu3

1School of Electrical Engineering, Henan University of Technology, Zhengzhou, Henan 450001, China
2College of Computer and Communication Engineering, Zhengzhou University of Light Industry, Zhengzhou, Henan 450002, China
3Marine Engineering Institute, Jimei University, Xiamen, Fujian 361021, China

Correspondence should be addressed to Tianfei Chen; chen_tianfei@163.com

Received 17 October 2017; Accepted 11 December 2017; Published 31 December 2017

Academic Editor: Shangguang Wang

Copyright © 2017 Tianfei Chen et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

To achieve fast calibration of line structured light sensor, a geometric calibration approach based on single circular calibration target is proposed. The proposed method uses the circular points to establish linear equations, and according to the angle constraint, the camera intrinsic parameters can be calculated through optimization. Then, the light plane calibration is accomplished in two steps. Firstly, when the vanishing lines of target plane at various postures are obtained, the intersections between vanishing lines and laser stripe can be computed, and the normal vector of light plane can be calibrated via line fitting method using intersection points. After that, the distance from the origin of camera coordinate system to the light plane can be derived based on the model of perspective-three-point. The actual experimental result shows that this calibration method has high accuracy, its average measuring accuracy is 0.0451 mm, and relative error is 0.2314%. In addition, the entire calibration process has no complex operations. It is simple, convenient, and suitable for calibration on sites.

1. Introduction

As a typical representative of noncontact vision measurement technology, the line structured light sensor has a wide application prospect in industry for dimensional analysis, on-line inspection, component quality, and reverse engineering, due to its simple structure, moderate accuracy, fast speed, large amount of information, and other advantage factors [1–6]. Generally, the sensor consists of one camera and one laser generator, and the laser triangulation is treated as basic principle, so the laser generator is displaced relative to the camera in space. When the structure is fixed, the task of sensor is to acquire the three-dimensional (3D) characteristic information of profile of the measured object. The acquisition process is described as follows: laser plane from laser generator is modulated by the depth of the measured object, and the laser stripe is formed, which contains 3D information of the measured object. The camera captures the object containing the deformed laser stripe. Then, 3D dense world points are generated by sampling points on each light stripe in the CCD image, and the calculation of 3D world points is based on the mathematical model of the sensor. When the line structured light sensor has already been assembled, some unknown model parameters exist, which directly influence the calculation precision of 3D world points for the measured object. Therefore, the calibration of model parameters is one of the key links for line structured light sensor.

The parameters calibration for line structured light sensor incorporates two stages: stage one is camera calibration, and the other is the calibration of light plane equation. According to the dimension of calibration target, the existing calibration method can be divided into the categories as below: (1) 3D stereo target. The wire drawing calibration method [7] and the sawtooth calibration method [8] are the earliest proposed methods. It should be noted that camera calibration is not involved in these two methods, and some precise auxiliary equipment is still needed, which leads to the fact that the calibration cost is relatively higher. Besides, the calibration process is tedious and the accuracy is limited. On the foundation of the previous work, Huynh et al. [9] and Wei et al. [10] make full use of cross-ratio invariance and double cross-ratio invariance to compute calibration points, and the
calibration accuracy is improved significantly. However, 3D precision calibration target is still employed, and calibration cost is higher all the same. In addition, high quality images are difficult to be captured, due to illumination occlusion of 3D stereo target. (2) Planar target. The calibration method that used free-moving planar target has become commonly popular for line structured light sensor [11–15]. This method can not only obtain high quality calibration images, but also reduce the calibration cost. However, some repeated calculation from each position of planar target to camera is required, so the complexity of computation is higher. (3) One-dimensional (1D) target. Wei et al. [16] and Han et al. [17] have proposed calibration methods based on 1D target, which can expand the application field of line structured light sensor. (4) Self-calibration method. Active vision calibration method for line structured light sensor has been proposed in [18], which can achieve automatic calibration without target, but it does not have universal applicability.

A field geometric calibration method for line structured light sensor based on single circular target is proposed in this paper. The proposed method makes full use of geometric constraints of a single circle to complete the parameters calibration of camera and light plane at the same time. The adopted single circular target has the advantages of simple structure and easy processing, and it can be placed free-movingly to capture more higher quality calibration images. In addition, the calculation process is not complicated or repeated, the algorithm is simple and fast, and it is flexible, convenient, and suitable for calibration on sites.

2. Mathematical Model of Line Structured Light Sensor

Figure 1 shows the mathematical model of line structured light sensor. In Figure 1, O_sX_sY_sZ_s represents camera coordinate system, and O_u-v is image plane coordinate system of CCD. O_sX_sY_s is defined as normalized plane coordinate system, and its distance to the origin of camera coordinate system is 1. Besides, O_sX_s∥O_cX_c, Y_s∥O_cY_c. The optical axis of camera O_cZ_c is perpendicular to normalized plane and CCD image plane. P is a point on the laser stripe, and p', p'' are, respectively, corresponding image points on CCD image plane and normalized plane.

Note that the homogeneous coordinates of point P in camera coordinate system are denoted by \( \mathbf{P} = (x_c, y_c, z_c, 1)^T \), and the homogeneous coordinates of its corresponding image points \( \mathbf{p}', \mathbf{p}'' \) in their coordinate system are, respectively, \( \mathbf{p}' (u_c, v_c, 1)^T \), \( \mathbf{p}'' (u_n, v_n, 1)^T \). Based on the perspective projection transformation, the relationships are described as follows:

\[
\begin{align*}
\mathbf{z}_c \cdot \mathbf{p}' & = [\mathbf{E} \ 0] \cdot \mathbf{P}, \\
\mathbf{p}' & = \mathbf{K} \cdot \mathbf{p}''.
\end{align*}
\]

\( \mathbf{E} \) represents a unit matrix of 3 * 3, and the homogeneous coordinate point \( \mathbf{p}'' \) can be treated as 3D coordinates of image point \( \mathbf{p}'' \) in the camera coordinate system. \( \mathbf{K} \) is camera intrinsic parameter matrix; its form is represented as follows:

\[
\mathbf{K} = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}.
\]

\( f_u, f_v \) are, respectively, the scale factors of the CCD image plane in the \( u, v \) axis direction. \( (u_0, v_0) \) represent the principal point coordinates of the camera; \( s \) is skew factor.

If the radial distortion of camera lens is further considered, the first-order model is commonly chosen to handle the nonlinear distortion effects, due to the fact that too many distortion parameters will make solution unstable [19]. Note that the actual corresponding image point coordinate of points \( \mathbf{p}' \) is \( (u_d, v_d)^T \); we have the relationship as follows:

\[
\begin{align*}
u_c & = (u_d - u_0) \cdot (1 + k \cdot r_d^2) + u_0, \\
v_c & = (v_d - v_0) \cdot (1 + k \cdot r_d^2) + v_0.
\end{align*}
\]

\( k \) is distortion parameter; \( r_d = \sqrt{(u_d - u_0)^2 + (v_d - v_0)^2} \).

Thus, the whole intrinsic camera parameters consist of the matrix \( \mathbf{K} \) and the distortion parameter \( k \).

Assume that the plane equation in camera coordinate system is expressed as follows:

\[
a_1 \cdot X + a_2 \cdot Y + a_3 \cdot Z + a_4 = 0.
\]

\( P \) is a point on the light plane, so its coordinate also satisfies constraint of (4). \( (a_1, a_2, a_3, a_4) \) are light plane parameters of sensor. If the camera intrinsic parameters and light plane parameters are known, the 3D coordinates of point \( \mathbf{P} \) on the light plane can be determined according to (1), (3), and (4).

3. Calibration of Sensor Parameters

3.1. Geometric Theorem. A single circular calibration target is used in this paper. In order to describe the proposed algorithm conveniently, some related geometric theorems are firstly introduced. Based on the geometric constraints of single circle, camera parameters and light plane parameters can be calibrated.

Figure 2 shows schematic diagram of single circular target and its imaging. In Figure 2(a), O is circle center, and A, B, C, D are intersection points between the circle and the line \( \overline{AB}, \overline{CD} \) across the circle center O. The angle between \( \overline{AB}, \overline{CD} \) is \( \theta \). \( L_{\infty} \) indicates the line at infinity, which is the intersection line formed by target plane and the plane at infinity. \( P_1, P_2 \) are, respectively, intersection points between line \( \overline{AB}, \overline{CD} \) and \( L_{\infty} \). \( m_1, m_2 \) are circular points of line \( L_{\infty} \). In Figure 2(b), vanishing points \( P_1, P_2 \) are, respectively, imaging points of \( P_1, P_2 \). Vanishing line \( l_{\infty} \) is the projection of \( L_{\infty} \) on CCD. In camera coordinate system, the direction of space straight line \( \overline{AB} \) is \( d \), the normal direction of target plane is \( n \), the homogeneous expressions of vanishing point \( P_1 \) and vanishing line \( l_{\infty} \), are, respectively, \( v, \overline{I} \). Based on the
3.2. Camera Calibration. The single circular target images under different postures are captured within the camera’s field of view. For each image, vanishing points for all straight lines across the circle center can be calculated according to property (1), and the vanishing line of target plane under this posture is obtained by fitting the straight line using vanishing points; then the imaging of circular points can be computed according to property (2). If the imaging coordinates of circular point at a certain position are \( I_m, J_m \), and its relationship with camera intrinsic matrix \( K \) can be expressed as

\[
I_m^T K^{-T} K^{-1} I_m = 0; \quad J_m^T K^{-T} K^{-1} J_m = 0.
\]  

(7)

the linear equations of matrix \( K \) can be constructed by formula (7), the detailed procedure can be referred to in [21]. However, the above process does not take into account the lens distortion parameters \( k \), and the lens distortion cannot be neglected in the field of vision measurement. In this paper, the optimization objective function is established according to

\[
\bar{v} \equiv K \cdot d, \quad (5)
\]

\[
\bar{t} \equiv K^{-T} \cdot n. \quad (6)
\]

The symbol \( \equiv \) represents equality in case of a constant factor.
the known angle constraint of the straight lines on the target; the formula is expressed as follows:

\[ f(K^*, k^*) = \min_{i=1}^{M} \sum_{j=1}^{N} \left( \theta_{ij} - \hat{\theta}_{ij}(K, k) \right)^2. \] (8)

In the above formula, \( M \) is the number of target images, and \( N \) is the number of angles between straight lines across the circle center on the target. \( \theta_{ij} \) represents the known angles between straight lines, and \( \hat{\theta}_{ij} \) is the actual calculation initial value, which can be treated as a function of camera intrinsic parameters \( K \) and \( k \). First, the extracted image points are corrected by formula (3), and then the vanishing points are computed. Using formula (5), the direction of lines through the center intersects with the circle center on the target plane, and then the vanishing point calculation formula is given by formula (3), and then the vanishing points are computed.

3.3. Light Plane Calibration. From formula (4), \( a_1, a_2, a_3, a_4 \) are the light plane parameters of the sensor. At present, the general methods [7–11] are used to calibrate light plane by plane fitting after seeking the calibration points on the light plane. Considering geometric meaning of light plane, \( (a_1, a_2, a_3) \) indicates the normal vector of light plane, \( a_4 \) represents the distance information from the origin of the coordinate system to the light plane, and then single circular target is used to complete calibration in two steps in this paper.

Figure 3 is schematic diagram of calibration for normal vector of laser plane. \( l_o \) is laser stripe line on CCD image plane which is formed by modulation between light plane and target plane. The Steger method [22] is adopted to extract the center coordinates of laser stripe line, and this method is stable and robust, and its accuracy can be reached to subpixel level. \( p_1 \sim p_3 \) are vanishing points of straight lines across the circle center \( O \) on the target, and its corresponding image coordinates can be determined by property (1). \( a_i, b_i \) are defined as intersection points between circle and straight lines across the center \( O \), and \( o \) is image coordinates of circle center; then the vanishing points calculation formula is given as follows:

\[ p_i = \frac{-1 \cdot (b_i - o) \cdot a_i - (a_i - o) \cdot b_i}{-1 \cdot (b_i - o) - (a_i - o)}; \quad i = 1, \ldots, n. \] (9)

\( l_{io} \) represents corresponding vanishing line of target plane, and its mathematical equation can be obtained by plane fitting using vanishing points. Note that the homogeneous coordinates expression of \( l_{io} \) on the CCD image plane is \( (n_x, n_y, n_z) \), so the formula is given by

\[ f(n_x, n_y, n_z) = \min_{i=1}^{n} \frac{n_x \cdot p_{ix} + n_y \cdot p_{iy} + n_z}{\sqrt{n_x^2 + n_y^2}}. \] (10)

In the above formula, \( p_{ix}, p_{iy} \) respectively, indicate \( x, y \) components of \( i_{th} \) vanishing point. \( v_p \) is the intersection point between line \( l_{io} \) and line \( l_{io} \), and it represents the vanishing point of laser stripe line under this posture. Similarly, target plane’s posture can be changed repeatedly, vanishing points \( v_{p1} \sim v_{pn} \) which indicate different directions of laser stripe lines on the light plane can be obtained, and then vanishing line of light plane can be calculated by line fitting, and the fitting equation is similar to formula (10). Then, the normal vector of light plane in the camera coordinate system can be computed by formula (6).

Note that it is necessary to make the target plane and the CCD image plane incline to each other; that is to say, a certain angle between them should exists when calibrating camera and light plane. If the two planes are parallel, the projection of the target plane becomes into affine transformation, and then the vanishing line of the target plane cannot be determined.

Figure 4 shows the schematic diagram of calibration for distance information. \( O_t \) is the origin of the camera coordinate system, \( D \) is the distance from the origin to the light plane. \( n_z \) represents the normal vector of light plane in the camera coordinate system. On the target plane, the laser stripe line intersects with the circle at point \( A \), and the line across the point \( A \) and the center \( O \) intersects with the circle at point \( B \), and their corresponding image points on CCD plane are, respectively, \( a, o, b \). At this moment, the principle of perspective-three-point between points \( A, O, B \) and viewpoint \( O_t \) will be used. If the distance of \( AO, OB \) and the angle of \( \angle AO_t O, \angle BO_t O \) are known, the coordinates of points \( A, O, B \) can be calculated [23]. On the CCD image plane, the coordinates of points \( a, o, b \) can be extracted by image processing algorithm. If the camera intrinsic parameters have been calibrated, the angle \( \angle AO_t O, \angle BO_t O \) can also...
be computed using the coordinates of the imaging points. While the distance of $AO, OB$ is equal to the radius of the circle, the coordinates of point $A$ on the light plane, that is vector $O_A$, can be calculated based on the model of perspective-three-point. Thus, the distance from the origin of camera coordinate system to the light plane is as expressed as follows:

$$D = \frac{\|O_A \cdot n_L\|}{\|n_L\|}.$$  \hfill (11)

In order to reduce the influence of noise and other factors, the position of the target plane can be changed repeatedly and the average calculation is carried on; the formula is described as follows:

$$D = \frac{\sum_{i=1}^{\text{Max}} D_i}{\text{Max}}.$$  \hfill (12)

$D_i$ represents distance information at the $i_{th}$ position, Max is the total number of points, and then the distance parameter $a_i$ is computed as follows:

$$a_i = \begin{cases} D_i & O_A \cdot n_L < 0 \\ -D_i & O_A \cdot n_L > 0. \end{cases}$$  \hfill (13)

Till then, the model parameters of sensor have already been calibrated completely.

4. Experiment Results and Accuracy Analysis

As shown in Figure 5(a), the line structured light sensor is composed of WAT-5352EX2 camera (resolution: 768 * 576), fixed focus lens made by Computar, and laser line generator of 650 nm wavelength. Figure 5(b) shows the single circular target used in this paper, and the radius is 20 mm. In order to extract the intersection points conveniently, the target checkered with black and white is adopted, and each angle in the target is 30 degrees.

4.1. Calibration Experiment. Turn on the laser generator, and the single circular target is free-movingly placed at different position within the measurement space of the sensor to capture calibration images. Figure 6 shows a set of calibration images. In the calibration images, the intersection points between the circle and lines across the center can be extracted by using OpenCV functions, and the functions are based on the Harris corner detection principle, and its extraction accuracy can reach 0.1 pixels. In addition, [21] is used to locate the circle center in the calibration image.
First of all, camera calibration is carried on according to Section 3.2, and the results are shown in Table 1.

Figure 7 shows the circle center locations influenced by lens distortion in a calibration image. The solid points are the intersection points of every two straight lines through the circle center, and the circle indicates the location center. The low distortion lens is used in this experiment, and its distortion is relatively smaller. Before correction, the solid points are basically distributed in a pixel cell, and the location center is within the solid points. After correction, the distribution of solid points is further narrowed on the direction of axis v, and then the result demonstrates that the accuracy of circle center location can be further improved by distortion correction.

Extract the laser stripe line, and calculate the intersection points between laser stripe line and the vanishing line of the target, and then fit the straight line; the fitting line is the vanishing line of light plane in CCD. In Figure 8, the circles are the intersection points between laser stripe line and vanishing line of target plane, and the fitting line is vanishing line of light plane. The camera intrinsic parameters have already been calibrated, so the normal vector of light plane in the camera coordinate system can be calculated by formula (6). The result is computed as follows:

\[ V = [a_1, a_2, a_3]^T = [0.7070, 0.0699, 0.7038]^T. \]  

Figure 9 is a laser stripe image which satisfies the model of perspective-three-point. The star points and the cross points are the intersection points of laser stripe line and the fitting ellipse, and the circle represents the center. Then the model of perspective-three-point is constructed, and the distance information of the light plane is derived according to the content of Section 3.3. In order to reduce the influence of noise and other factors, more stripe images should be captured as shown in Figure 9, and then the average calculation is carried on after calculating distance information for each laser stripe image. By formula (13), the distance information of light plane is determined as \( a_4 = -128.1163 \). Finally, the calibration result of light plane equation is expressed as follows:

\[ 1.0045 \cdot X + 0.0993 \cdot Y + Z - 182.0375 = 0. \]  

### 4.2. Accuracy Analysis

In order to verify the accuracy of the proposed calibration method, a high precision planar target with solid circle is used. As shown in Figure 10, turn on the laser generator and capture the target image. In the local world coordinate system defined by planar target, the coordinates of feature points on the stripe line can be computed using principle of cross-ratio invariance, and then reference distance value can be calculated among the feature points. According to the calibration results of line structured light sensor, the distance among feature points can also be calculated; then compare them with the reference distance value. The comparative results are shown in Table 2.

From the analysis data in Table 2, we can see that the proposed method has higher calibration accuracy, the average absolute measurement deviation is 0.0451 mm, and the average relative measurement error is 0.2314%, which can meet the precision requirements of most application areas.

### 5. Conclusion

In this paper, a geometric calibration method for line structured light sensor is presented. This method uses single circular target to construct geometric constraints, and both camera intrinsic parameter and light plane equation can be calibrated at the same time. The single circular target’s structure is simple and easy to process. The proposed algorithm has no complex or repeated calculation process, and it is simple, fast, flexible, and suitable for field calibration. The experimental results demonstrate that this method has higher calibration accuracy, the average measurement error is 0.0451 mm, and

---

**Table 1: Camera intrinsic parameters.**

<table>
<thead>
<tr>
<th>No.</th>
<th>( f_a )</th>
<th>( f_u )</th>
<th>( s )</th>
<th>( u_0 )</th>
<th>( v_0 )</th>
<th>( k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1904.3</td>
<td>1907.6</td>
<td>8.8625</td>
<td>410.1198</td>
<td>251.8105</td>
<td>2.3001\times 10^{-4}</td>
</tr>
</tbody>
</table>

**Table 2: Measurement data by line structured light sensor.**

<table>
<thead>
<tr>
<th>No.</th>
<th>Reference Value/mm</th>
<th>Measurement Value/mm</th>
<th>Absolute Error/mm</th>
<th>Relative error/%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 2)</td>
<td>10.0167</td>
<td>9.9483</td>
<td>0.0685</td>
<td>0.6838</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>20.0148</td>
<td>19.9231</td>
<td>0.0917</td>
<td>0.4582</td>
</tr>
<tr>
<td>(1, 4)</td>
<td>35.0251</td>
<td>34.9509</td>
<td>0.0742</td>
<td>0.2188</td>
</tr>
<tr>
<td>(1, 5)</td>
<td>45.0532</td>
<td>44.9354</td>
<td>0.1178</td>
<td>0.2614</td>
</tr>
<tr>
<td>(1, 6)</td>
<td>55.0606</td>
<td>54.9890</td>
<td>0.0716</td>
<td>0.1300</td>
</tr>
<tr>
<td>(2, 3)</td>
<td>10.0018</td>
<td>9.9749</td>
<td>0.0270</td>
<td>0.2696</td>
</tr>
<tr>
<td>(2, 4)</td>
<td>25.0111</td>
<td>25.0026</td>
<td>0.0085</td>
<td>0.0339</td>
</tr>
<tr>
<td>(2, 5)</td>
<td>35.0370</td>
<td>34.9871</td>
<td>0.0498</td>
<td>0.1422</td>
</tr>
<tr>
<td>(2, 6)</td>
<td>45.0446</td>
<td>45.0408</td>
<td>0.0038</td>
<td>0.0085</td>
</tr>
<tr>
<td>(3, 4)</td>
<td>15.0102</td>
<td>15.0278</td>
<td>0.0175</td>
<td>0.1168</td>
</tr>
<tr>
<td>(3, 5)</td>
<td>25.0401</td>
<td>25.0123</td>
<td>0.0279</td>
<td>0.1114</td>
</tr>
<tr>
<td>(3, 6)</td>
<td>35.0469</td>
<td>35.0659</td>
<td>0.0190</td>
<td>0.0543</td>
</tr>
<tr>
<td>(4, 5)</td>
<td>10.0372</td>
<td>9.9845</td>
<td>0.0527</td>
<td>0.5247</td>
</tr>
<tr>
<td>(4, 6)</td>
<td>20.0395</td>
<td>20.0381</td>
<td>0.0014</td>
<td>0.0069</td>
</tr>
<tr>
<td>(5, 6)</td>
<td>10.0078</td>
<td>10.0537</td>
<td>0.0459</td>
<td>0.4582</td>
</tr>
</tbody>
</table>

Average error 0.0451 0.2314
the relative error is 0.2314%, which meets the requirements of the detection accuracy in most application areas.

**Conflicts of Interest**

The authors declare that they have no conflicts of interest.

**Acknowledgments**

This work was supported in part by the National Natural Science Foundation of China (U1604151, 61302118), Outstanding Talent Project of Science and Technology Innovation in Henan Province (174200510008), Program for Scientific and Technological Innovation Team in Universities of Henan Province (16IRTSTHN029), Program for Science and Technology Innovation Talents in Universities of Henan Province (17HASTIT022), the Funding Scheme of Young Key Teacher of Henan Province Universities (2016GGJS-087), and the Fundamental Research Funds of Henan University of Technology (2015QNJH13, 2016XTCX06).
References


