Hindawi
Mobile Information Systems
Volume 2018, Article ID 1904636, 12 pages
https://doi.org/10.1155/2018/1904636

Research Article

SRAF: A Service-Aware Resource Allocation Framework for VM Management in Mobile Data Networks

Kang Liu,[1] Ruijuan Zheng,[1] Mingchuan Zhang,[1] Chao Han,[2] Junlong Zhu,[1] and Qingtao Wu[1]

[1]*College of Information Engineering, Henan University of Science and Technology, Luoyang 471000, China*
[2]*Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China*

Correspondence should be addressed to Ruijuan Zheng; zhengruijuan@haust.edu.cn

Received 11 September 2018; Accepted 25 October 2018; Published 2 December 2018

Academic Editor: Wenchi Cheng

Copyright © 2018 Kang 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.

Service latency and resource utilization are the key factors which limit the development of mobile data networks. To this end, we present a service-aware resource allocation framework, called SRAF, to allocate the basic resources by managing virtual machine (VM). In SRAF, we design two new methods for better virtual machine (VM) management. Firstly, we propose the self-learning classification algorithm (SCA) which executes the service request classification. Then, we use the classification results to schedule different types of VMs. Secondly, we design a sharing mode to jointly execute service requests, which can share the CPU and bandwidth simultaneously. In order to enhance the utilization of resources with the sharing mode, we also design two scaling algorithms, i.e., the horizontal scaling and the vertical scaling, which execute the operation of resource-level scaling and VM-level scaling, respectively. Furthermore, to enhance the stability of SRAF and avoid the frequent operation of scaling, we introduce a Markov decision process (MDP) to control VM migration. The experimental results reveal that SRAF greatly reduces service latency and enhances resource utilization. In addition, SRAF also has a good performance on stability and robustness for different situations of congestion.

1. Introduction

Virtual machine (VM) management based on service awareness is a new method which can greatly reduce the service latency and enhance the resource utilization. In addition, VM management has been widely used in various mobile data networks, such as information center network (ICN), mobile vehicle network (MVN), and mobile cloud network (MCN). However, the resource pools of networks are limited. Moreover, the service latency and resource utilization are interactional. Therefore, how to reduce the service latency and how to enhance the utilization of resources, simultaneously, have been the focus point of research studies, especially for MCN [1, 2].

For this reason, service latency and resource utilization have become main aspects in many research studies. In the perspective of service latency, Reference [3] proposes the Predictable Resource Guarantee Scheduler scheme to realize the proportional sharing of CPU and I/O bandwidth, which reduces the waiting time in the Xen platform. Reference [2] uses the cloudlet selection strategy to schedule the cloudlets for cutting down the response time. In addition, there are also some studies using the method of cutting down the distance between locations to reduce the latency. For example, Reference [1] aims to find the shortest path between the user and the nearest cloud datacenter for reducing the transmission latency. For reducing the queueing time of requests, Reference [4] uses the method of active communication between controllers to proactively pull the requests when the controller finishes its requests so as to cut down the queue length. On the contrary, in the perspective of resource utilization, Reference [5] presents a smart migration mechanism to implement processor memory optimization based on VM placement. References [6, 7], respectively, use the methods of Bejo and kNN classification schemes to classify the requests for better scheduling. Reference [8] designs a Lyapunov optimization framework to improve the efficiency of the mobile-edge computing. The purpose of the

Lyapunov optimization framework is to minimize the resource overload by VM scheduling.

The works cited above propose many new ideas or methods to realize the optimal request schedule or optimal resource configuration, which can reduce delay or enhance the resource utilization. However, due to the complexity of the cloud network and diversity of mobile devices, the requests are also various and uncontrolled. So, the simple objective studies do not always have a good performance on different factors because many factors are interactional. Therefore, there are some researchers who design efficient methods with an overall framework to optimize these problems. In [9], the authors design a resource sharing framework named "Symbiosis" to realize the sharing of CPU and bandwidth. When one request is working in the CPU, the Symbiosis will make another request to perform the transmission. Moreover, the Symbiosis can efficiently reduce the service latency of the requests. In [6], the authors propose a new classification algorithm named "Bejo" to classify the requests. The classification results are used to perform the VM scheduling. The fitting VM for requests can enhance the utilization of resources. Therefore, we propose a new framework called SRAF to execute requests classification and resources sharing based on the strength of research studies in [6, 9].

For measuring the performance of SRAF, we analyze MCN in detail. The SRAF can be divided into two aspects. Firstly, we propose a self-learning classification algorithm (SCA) to perform the classification operation before the request is sent to the VM. The SCA is designed by two weighting methods, location weighting and feature weighting [10], which can improve the veracity of requests classification. The precise classification results can help the request find a fitting VM so as to reduce the service latency. Secondly, we design a sharing mode (Figure 1) to realize the resource sharing in a VM. Furthermore, in order to improve the utilization of the resources, we also design two VM scaling algorithms, the horizontal scaling and vertical scaling. The former is to realize the resource-level scaling in a VM. When the utilization of CPU or bandwidth is too high, the algorithm will add corresponding resources to the VM for avoiding overload, or otherwise for scaling down. The latter is to perform the VM-level scaling. When all the VMs are busy and the arrival tasks are continuously growing, new VMs are created, or otherwise released.

The contributions of this paper are as follows:

(i) We propose a new framework named "SRAF" to improve the resource utilization and reduce the service latency simultaneously.

(ii) We design the SCA which has the self-learning capacity for updating features so as to classify the service requests. In addition, SCA can improve the accuracy of classification continuously until all the features are learned.

(iii) We introduce a Markov decision process (MDP) to control VM migration so as to reduce the frequent scaling operation and enhance the stability of SRAF.

(iv) We propose a Combination Scheduling Cost Model and a sharing mode for mobile data networks. Combination Scheduling Cost Model can systematically operate VMs scheduling and scaling. Moreover, the resource utilization is improved directly via the sharing mode.

The rest of this paper is organized as follows: Section 2 has a brief introduction of related research works. We particularly introduce the details of each component of SRAF in Section 3. Section 4 shows the overall process of SRAF, including each model and related algorithm. Section 5 presents the feasibility and performance of SRAF by some experiments. Section 6 presents a brief conclusion of this paper.

2. Related Work

To reduce the service delay and enhance the utilization of resources, many studies propose various methods. For example, Reference [3] proposes a new prediction method to reach the objective of resource sharing. Researchers use the prediction method to predict the demand of the next duration time so as to adjust the resources for enhancing the utilization of resources. References [2, 5] use different methods to schedule the cloudlet and VM for reducing the response time and free time of resources, respectively. Due to the diversity of mobile access devices, the requests are also different and uncontrolled. For this reason, some studies design different classification algorithms to classify different requests, which depend on demand or input data for prior disposal [6, 7]. Then, researchers can use some classical and effective methods to reduce the queueing time and improve the utilization of resources, such as shortest job first (SJF) [12] and priority-aware longest job first (PA-LJF) [13]. Moreover, there are also some other new improved methods, such as shortest expected-remaining service time policy (SE-RSTP) [14] and dynamic-threshold service policy [15], which have a better performance for reducing delay and improving the QoS.

In addition, due to the randomness of service requests, some researchers use the Markov decision process (MDP) to quantify the overall process of cloud service. For example, Reference [16] uses the dynamic Markov decision process to model the process of VM scheduling. Then, the value iteration algorithm is used to find the optimal VM control policy for reducing energy expenditure. Reference [17] also uses the MDP to quantify the overall VM control. It uses the Bellman optimality equation to find a global optimum threshold so as to cut down erratic operation of VMs. To enhance the veracity of task scheduling by MDP, Reference [18] designs a semi-Markov decision process to select some computation-intensive tasks for offloading so as to reduce the computations in mobile devices.

Due to the above analysis, single method or policy can only realize one or two objectives. Therefore, some researchers choose to design an overall framework to handle multiple objectives. For example, References [8, 9] design different frameworks, Lyapunov optimization framework

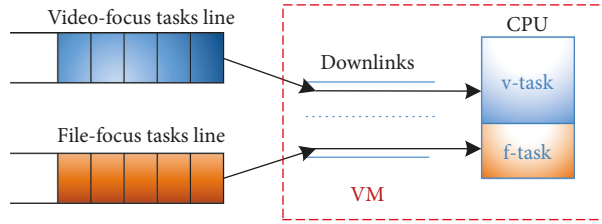


Figure 1: The sharing mode of the high-type VMs.

and Symbiosis, to control the scheduling of requests and VMs, respectively. Furthermore, there are also some researchers who add classification methods to enhance the performance of different frameworks for different objectives. For example, Reference [19] uses the reweighting method to label different factors by machine learning. Reference [11] uses the multi-instance learning method (MIL) to quantify different data for precise classification with the probabilistic graphical framework. Reference [10] uses the method of local feature selection to classify the data directly. In addition, many studies also use the sharing method to improve the utilization of resources in different fields. For example, Reference [20] proposes a feasible and truthful incentive mechanism (TIM) to realize the resource sharing with the trade-off between users and service providers. Reference [21] uses the sharing mode to satisfy the resources demand of the remote radio network and central virtual base station so as to maximize downlink of networks.

3. Architecture Design

In this section, we designed a green cloud resource allocation framework called SRAF. The objective is to reduce the delay of scheduling the service request and improve the resource utilization simultaneously. Our framework contains three layers, the User Layer, the Request Manager Layer, and the Resource Provider Layer.

In Figure 2, we show the overall response process of service requests. The User Layer has many users with various mobile terminals which send service requests to the mobile cloud. The Request Manager Layer is the most important layer to receive the service requests from the User Layer. Its main duty is to make optimization management of service requests and VMs. Then, the results are sent to the Resource Provider Layer for VM configuration. The Resource Provider Layer provides basic resources for the service. The Request Manager Layer includes four components:

(1) The History Loads is used to store the requests and their categories which can help the Classification Manager in updating its feature mapping library. The category information comes from the Combination Scheduling Manager when requests are serviced.

(2) The Classification Manager analyzes the information from the requests and classifies the requests into three types, i.e., file-focus tasks, video-focus tasks, and normal tasks, depending on the demand of bandwidth and CPU resources (details are given in Section 4.2).

(3) The Combination Scheduling Manager uses the classification results from the Classification Manager to make combination scheduling of the requests; then, it performs resource allocation and pushes the real information of the requests to the History Loads for updating its features (details are given in Section 4.4) during the operation.

(4) The Monitor Manager monitors the utilizations of the CPU and bandwidth of each active VM to support real-time service information.

The Resource Provider Layer has many resource pools, such as CPU, bandwidth, and memory. This layer provides basic resources for VMs so as to handle these service requests.

4. Model Design and Algorithm Analysis

4.1. System Model. Our goal is to reduce the service delay and enhance the resource utilization by the proposed system architecture, which can choose a suitable VM in the physical machine for the service requests. In other words, we will make a fitting combination of request, VM, and physical host, described as $\langle \text{task}_i \rangle \longrightarrow \langle \text{VM}_j \rangle \longrightarrow \langle \text{host}_k \rangle$, $i \in \{1, 2, \ldots, M\}$, $j \in \{1, 2, \ldots, N\}$, and $k \in \{1, 2, \ldots, Q\}$.

Definition 1. For a set of hosts $\langle \text{host}_k \rangle = \{\text{host}_1, \text{host}_2, \ldots, \text{host}_k, \ldots, \text{host}_Q\}$ and the VM set $\langle \text{VM}_j = \{\text{VM}_1, \text{VM}_2, \ldots, \text{VM}_j, \ldots, \text{VM}_N\}$, the connection of them can be defined as a matrix $U_{Q\times N}$, i.e.,

$$U_{Q\times N} = \left[u_{kj}\right]^{Q\times N}, \tag{1}$$

where if VM_j is not created or released on host_k at the beginning, then we set $u_{kj} = -1$. If VM_j locates on host_k, then $u_{kj} \in (0, 1)$. At the same time, we use the $u_{kj}^{\text{cpu}} \in (0, 1)$ to denote the utilization of CPU on VM_j and use $u_{kj}^{\text{bw}} \in (0, 1)$ to denote the utilization of bandwidth on VM_j.

Definition 2. For a set of tasks $\langle \text{task}_i \rangle = \{\text{task}_1, \text{task}_2, \ldots, \text{task}_i, \ldots, \text{task}_M\}$ and a virtual machine set $\langle \text{VM}_j \rangle = \{\text{VM}_1, \text{VM}_2, \ldots, \text{VM}_j, \ldots, \text{VM}_N\}$, the distribution of the tasks is defined as a matrix $A_{M\times N}$, i.e.,

$$A_{M\times N} = \left[a_{ij}\right]^{M\times N}, \tag{2}$$

where $a_{ij} \in \{0, 1\}$; if $a_{ij} = 1$, then that task_i is distributed on VM_j. If $a_{ij} = 0$, there is no connection between task_i and VM_j.

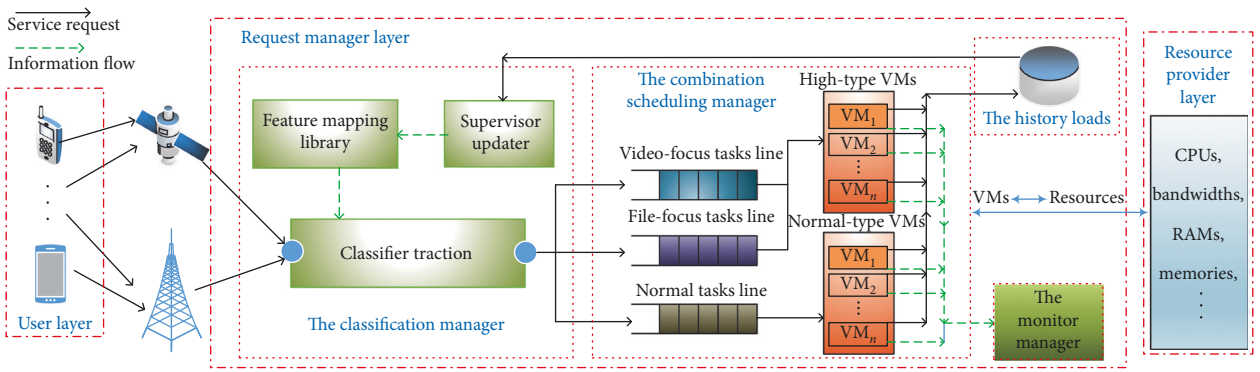


Figure 2: Architecture of the SRAF framework.

4.2. Classification Model. Considering the complexity of mobile terminals, we will make the classification from the perspective of service in the mobile cloud network. Therefore, all of the requests can be divided into three kinds of tasks, i.e., the video-focus tasks, file-focus tasks, and normal tasks.

The classification model has three components: (1) *feature mapping library* is used to store the relationship of the features and their corresponding service classes so as to make classification; (2) *classifier traction* is used to classify the tasks according to their input data and the mapping in the feature mapping library; and (3) *supervisor updater* is used to supervise and update the mapping in the feature mapping library based on the information from the History Loads.

In this section, we use self-learning classification algorithm (SCA) to operate classification. SCA is designed by the machine learning technology [22, 23] because we can constantly extend and update the feature mapping library so as to improve the accuracy of the classification by the machine learning. The SCA also improves the traditional classification algorithm by multiple weighting and uses the semisupervised method to update the features in the feature mapping library according to the feedback from the History Loads [24]. SCA uses the method of learning to expand the new relationship of features and service requests so as to enhance the veracity of classification results. Therefore, SCA uses the combination of location weighting, feature weighting, and self-learning methods to determine the final class of each request.

In the process of SCA, we use G_V and G_F to denote the mapping set of videos and files, respectively. We firstly append some typical features into the feature mapping library for the mapping set $G = \{G_V, G_F\}$, such as $G_V = \{\langle tv, video\rangle, \langle dvd, video\rangle, \ldots, \langle avi, video\rangle\}$ and $G_F = \{\langle doc, file\rangle, \langle wps, file\rangle, \ldots, \langle ppt, file\rangle\}$. Then, SCA uses the input data of requests to find the mapping in the G. In the process of mapping, we use the methods of location weighting and feature weighting to ensure the request has a precise classification. Feature weighting is that different features have different weighting. For example, the feature "video" has a large weighting than "avi" for indicating video tasks. Location weighting is that we use the location of different features in the URL to weighting. For example, if a request URL is divided into n segments, then we can use these n segments to form a one-dimensional array $L = \{l_1, l_2, \ldots, l_n\}$. We use α_i to denote the location weight of the $i-$th feature:

$$\alpha_i = \frac{(n - n_{loc})}{n}, \tag{3}$$

where n_{loc} is the location of the i–th feature in the order. Moreover, the more forward the location is in the order, the more important the feature is in the description [25]. We use β as the final weight to determine which task line the request should be scheduled. For example, there is a request with some features in G_V, such as $fea = \{k_i \in L \mid i = 1, 2, \ldots, m, m < n\}$ and $fea \subseteq G_V$. The weighting of k_i is β_v^i for $i \in \{1, 2, \ldots, m\}$. Hence, the total weighting of video features in the request URL is calculated by

$$\beta_v = \sum_{i=1}^{m} \alpha_i \beta_v^i. \tag{4}$$

Similarly, the total weighting of file features in the request URL is calculated by

$$\beta_f = \sum_{i=1}^{m} \alpha_i \beta_f^i. \tag{5}$$

Finally, the attribution of the service request is calculated by

$$\beta = \max\{\beta_v, \beta_f\}. \tag{6}$$

If $\beta_v \geq \beta_f$, this request is transmitted to the video-focus tasks line. Otherwise, this request is transmitted to the file-focus tasks line. If β_v and β_f are equal to zero, this request will be transmitted to the normal tasks line. The process of SCA is shown in Algorithm 1.

4.3. VM Migration. According to Definition 1 and Definition 2, the overall process of SRAF is to find an appropriate location in VM_j and $host_k$ for $task_i$. If we define a location function as $B(t) := [a_{ij}(t), u_{kj}(t)]$, then that $task_i$ is scheduled on VM_j and VM_j located in $host_k$ at time t. In addition, when $task_i$ is classified by classifier traction, it will be scheduled to VM_j and it cannot be transferred. Therefore, when the resource of $host_k$ cannot satisfy the demand of

Input: request information
Output: line to which request belongs
(1) Initialize G_V, G_F
(2) Divide the URL into L;
(3) Calculate β_v by circularly comparing the features in L, G_V according to (3) and (4)
(4) Calculate β_v by circularly comparing the features in L, G_F following (3) and (5)
(5) Calculate β by (6);
(6) Output the attribution of the request.

Algorithm 1: The description of SCA.

VM_j, VM_j will be migrated to another host at timeslot t based on practical conditions from the Monitor Manager. Let $D(t) \coloneqq \{d_{j,kk'}(t) \mid j \in N; k, k' \in Q\}$ represents the set of action, where $d_{j,kk'}(t)$ means VM_j can migrate from $host_k$ to $host_{k'}$ at timeslot t. Correspondingly, each $d_{j,kk'}(t)$ has a migration probability as $p_{j,kk'}(t)$, and all the probabilities make up probability set $\mathscr{P}$, indexed as

$$\mathscr{P}(t) \coloneqq \{p_{j,kk'}(t) | j \in N; k, k' \in Q\}, \quad \sum_{k,k' \in Q} p_{j,kk'}(t) = 1. \tag{7}$$

The cost function of VM migration is defined as

$$f_j(t) = \sum_{k,k' \in Q} C_{k,k,} p_{j,kk'}(t), \tag{8}$$

which means the additional expenditure of VM migration. $C_{k,k'}$ is the migration expenditure of VM_j from $host_k$ to $host_{k'}$. In addition, $C_{k,k'}$ is influenced by the migration distance and the latter operation expenditure. Hence, let $E = \{B(t), D(t), \mathscr{P}(t), f_j(t)\}$ be a basic MDP to represent VM migration because the arrival of service requests is based on the Poisson process [32, 33]. So, if the capacity of VM_j is stationary, the overload and VM migration will be a loop in a long time. Therefore, we can get a stationary policy π to control the overall process of VM migration. Now, we use the Bellman optimality equation and the method of dynamic programming to obtain the optimal control policy π [26, 27]. We introduce the state value function as follows:

$$V_T^{\pi}(b(t)) = \sum_{d_{j,kk'}(t) \in D(t)} \pi(kk') \cdot \sum_{k,k' \in Q} p_{j,kk'}(t) \cdot \left(\frac{1}{T} R\left(d_{j,kk'}(t)\right) + \frac{T-1}{T} V_{T-1}^{\pi}(b(t-1))\right), \tag{9}$$

where $R(d_{j,kk'}(t))$ means that the penalty of VM_j operates the action $d_{j,kk'}(t)$; T is the discount factor to determine the importance of history data; $b(t)$ is the location of VM_j at timeslot t in B(t); and $\pi(kk')$ means that VM migrates from $host_k$ to $host_{k'}$ by policy π. We use (9) to select the optimal state at the next timeslot so as to maximize the reward. Then, the action value function is

$$P_T^{\pi}\left(b(t), d_{j,kk'}(t)\right) = \sum_{k,k' \in Q} p_{j,kk'}(t)\left(\frac{1}{T} R\left(d_{j,kk'}(t)\right) + \frac{T-1}{T} V_{T-1}^{\pi}(b(t-1))\right). \tag{10}$$

We use (10) to determine the action which can satisfy the optimal state at the next timeslot. Finally, we use the value iteration algorithm to handle the control policy π [27–29].

In Algorithm 2, we aim to get and update the control policy π, which can control VM migration based on history data in the History Loads. The control policy π can improve the resource utilization and load balancing in the system. The overall process of Algorithm 2 is to update the state value function by finding the optimal reward path. In other words, we need to traverse all $b(t)$ and choose an optimal location to migrate VM, which can maximize the reward of all the VMs. Then, we find an optimal string of states of $B(t)$ over time. Finally, the algorithm uses the backstepping approach to get control π using the action value function and the optimal string of states in $B(t)$.

4.4. Combination Scheduling Cost Model. We firstly design two types of VMs, the high type and the normal type, which have a different capacity of conducting tasks. The tasks in III-B are divided into three kinds. We use the sharing mode (Figure 1) and Symbiosis in [9] to execute the scheduling according to the tasks' demand for resources.

From Figure 2, we propose a sharing mode for the video-focus tasks and file-focus tasks to jointly share the resources of the high-type VMs. With the sharing mode, one VM can simultaneously execute two tasks, one video-focus task and one file-focus task. Moreover, two tasks share the resources of their owner VM, such as the bandwidth and CPU resources. If the video-focus tasks are less than the file-focus tasks, the VM can have two file-focus tasks. If the video-focus tasks are more than file-focus tasks, the VM will execute one video-focus task and wait for the file-focus tasks. On the contrary, the normal tasks are processed on normal-type VMs by the Symbiosis [9] based on the idea of space sharing in CloudSim [30].

In Figure 3, we use an example to show the process of sharing mode. We design three tasks and give different

Input: $E = \{B(t), D(t), \mathscr{P}(t), f_j(t)\}$, T, θ
Output: π
(1) $\forall b(t) \in B(t)$, $V(b(t)) = 0$.
(2) for $t = 1, 2, 3, \ldots$ do
(3) $\forall b(t) \in B(t), \forall \mathrm{VM}_j \in \langle \mathrm{VM}_j \rangle : V'(b(t)) = \max_{d_{kk'}(t)} \sum_{b(t) \in B(t)} p_{j,kk'}(t) \cdot (1/TR(d_{j,kk'}(t)) + T - 1/TV^{\pi}_{T-1}(b(t-1)))$
(4) if $\max_{b(t) \in B(t)} |V(b(t)) - V'(b(t))| < \theta$ then
(5) break.
(6) else
(7) $V = V'$.
(8) end if
(9) end for
(10) Output $\pi = \mathrm{argmax}_{b(t) \in B(t)} P^{\pi}_{T}(b(t), d_{kk'})$.

Algorithm 2: Computing π by value iteration.

lengths for each task in transmission by bandwidth and the execution length of CPU. In order to express clearly, we set the execution efficiency of bandwidth as 2 in one interval and that of CPU as 3 in one interval. Due to the sharing mode, one VM can have two tasks, and these tasks share resources based on the percentage of 1:1. So, the process of working is as follows: at time 0, the v-task1 and f-task1 were allocated to the same VM. Firstly, v-task1 and f-task1 begin to transfer their transmission length (T-length) by sharing the bandwidth. At time 2, the T-length of f-task1 is finished. The f-task1 begins to solely execute its execution length (E-length) on CPU. At the same time, v-task1 occupied the bandwidth by itself for transferring its remaining T-length. At time 3, v-task1 finishes its T-length and begins to execute its E-length by sharing with f-task1. At time 5, f-task1 finishes the E-length and leaves the VM. Then, the f-task2 begins to transfer for working. So, the f-task2 uses bandwidth by itself, and v-task1 also uses the CPU by itself. At time 6, the f-task2 finishes its T-length and begins to occupy the CPU with v-task1. At time 12, f-task2 is finished and leaves the VM. V-task1 uses CPU by itself. Finally, v-task1 finishes its work and leaves the VM at time 14. At this point, the overall process is finished. Algorithms 3–5

In the following, we present the overall algorithm process of SRAF. When the system scheduling algorithm (SSA) starts, we will create basic VMs (line 6) for firstly scheduling. The Monitor will constantly monitor $u^{\mathrm{bw}}_{kj}(s)$ and $u^{\mathrm{cpu}}_{kj}(s)$ of every VM at the beginning of the s – th interval Δt. Then, the algorithm will choose an operation by the control policy π, whether doing VM migration or VM scaling, for minimizing the cost (lines 7–11). When the task$_i$ arrives, SSA will classify task$_i$ into the corresponding task line (lines 1–4 and 13). When the task lines have tasks, we will schedule them according to $u^{\mathrm{bw}}_{kj}(s)$ and $u^{\mathrm{cpu}}_{kj}(s)$ so that we can make full use of the resources (lines 14–17).

In the process of horizontal-scaling algorithm, we set the CPU maximum utilization threshold and the bandwidth maximum utilization threshold as up_{cpu} and up_{bw}, respectively. Then, we use up_{cpu} and up_{bw} to compare with $u^{\mathrm{cpu}}_{kj}(s)$and $u^{\mathrm{bw}}_{kj}(s)$ at the beginning of each interval Δt, respectively. Due to the comparison results, the algorithm chooses to perform different operations of resource-level scaling of every active VM.

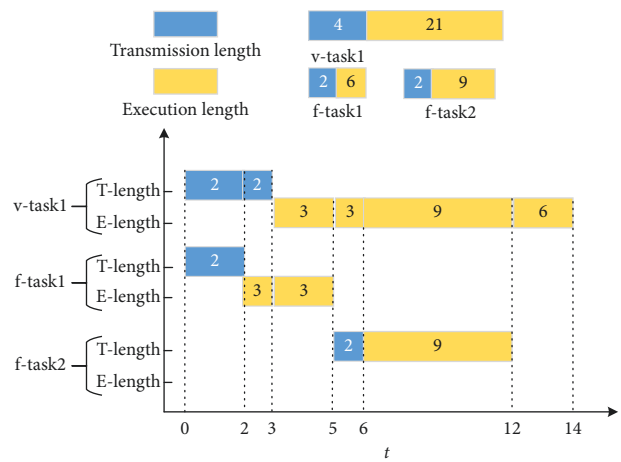


Figure 3: An example for the sharing mode.

In Algorithm 5, the arrival ratio and finished ratio represent the quantity of the arriving and finished requests at the beginning of the interval Δt, respectively. Its main duty is to control VMs in the overall framework. According to the situation of resource utilization and the quantity of requests, the algorithm executes the VM-level scaling.

5. Performance Evaluation

To evaluate the proposed framework in this paper, we build the SRAF in CloudSim which is a discrete event simulator [30]. In CloudSim, we can make duplicate and controllable experiments following our idea. CloudSim can support various environments for the resource allocation and scheduling study. We implement all the models and algorithms in CloudSim for comprehensive evaluation and analysis.

In the following, we show the overall calculation process of the execution cost for measuring performance. In the process of scheduling, we can use (11) and (12) to quantify the cost of each VM at the s – th interval Δt:

(1) procedure CLASSIFICATION METHOD (task$_i$)
(2) Uses the **SCA** in Algorithm 1 to classify task$_i$;
(3) Puts the task$_i$ into the corresponding line;
(4) **End**
(5) **procedure MAIN**
(6) Initializes basic VMs into two types, the high-type VM line and the normal-type line;
(7) **while** the system is running and in the beginning of an interval **do**
(8) Monitors $u_{kj}^{\mathrm{bw}}(s)$, $u_{kj}^{\mathrm{cpu}}(s)$ of every VM;
(9) Calls Algorithm 2 to determine the VM migration.
(10) Calls **Horizontal-scaling algorithm**; //see Algorithm 4
(11) Calls **Vertical-scaling algorithm**; //see Algorithm 5
(12) **end while**
(13) **while** (task$_i$ is coming) **do**
(14) **CLASSIFICATION METHOD** (task$_i$);
(15) **while** there have tasks in the three lines **do**
(16) Schedules the tasks from the video-focus tasks line and file-focus tasks line on the high-type VM by the sharing mode;
(17) Schedules the normal task into normal-type VM;
(18) **end while**
(19) **end while**
(18) **End**

Algorithm 3: System scheduling algorithm.

(1) Begin
(2) **if** $u_{kj}^{\mathrm{cpu}}(s) > up_{\mathrm{cpu}}$ and $host_k$ has enough CPU resource **then**
(3) Scales up the CPU resource of VM$_j$;
(4) **end if**
(5) **if** $u_{kj}^{\mathrm{bw}}(s) > up_{\mathrm{bw}}$ and $host_k$ has enough bandwidth resource **then**
(6) Scales up the bandwidth resource of VM$_j$;
(7) **end if**
(8) **if** VM$_j$ is idle for a long period **then**
(9) Executes the resource scale down;
(10) **end if**
(11) **End**

Algorithm 4: Horizontal-scaling algorithm.

(1) Begin
(2) **if** all the VMs are busy and arrival ratio > finished ratio **then**
(3) Creates new VM;
(4) **else**
(5) Releases the idle VM;
(6) **end if**
(7) **End**

Algorithm 5: Vertical-scaling algorithm.

$$C_{kj}^{q}(s) = C_{\mathrm{bas}}^{q} + \delta^{q} \cdot \Delta t \cdot u_{\mathrm{kj}}^{q}(s), \tag{11}$$

$$g_{kj}^{q}(s) = C_{kj}^{q}(s) + C_{\mathrm{sca}}^{q}(s), \tag{12}$$

where $s \in \{1, 2, \ldots, T_j\}$, in which T_j is the overall execution time of VM$_j$; $q \in \{\mathrm{cpu}, \mathrm{bw}\}$, in which cpu represents the CPU resource and bw represents the bandwidth resource; $u_{kj}^{q}(s)$ contains two aspects, i.e., $u_{kj}^{\mathrm{cpu}}(s)$ and $u_{kj}^{\mathrm{bw}}(s)$, which represent the utilization of CPU and bandwidth of VM$_j$ at the s – th interval Δt, respectively; $C_{kj}^{q}(s)$ is the ordinary cost of q when the VM$_j$ has tasks and is working; C_{bas}^{q} is the basic cost of creating the VM on the elements of set q; $C_{\mathrm{sca}}^{q}(s)$ is the scaling cost of q when the VM performs the scaling operation at the s – th interval Δt; and δ^{q} is the execution cost of one interval Δt on the elements of set q. Furthermore, the

practical resources cost of all the VMs on host_k in their working period is given by

$$g_{\text{total}} = \sum_{j=1}^{N} \sum_{s=1}^{T_j} g_{kj}^{q}(s). \tag{13}$$

The total cost of all the resources in VMs is given by

$$g_{\text{all}} = \sum_{j=1}^{N} \sum_{s=1}^{T_j} \left(C_{\text{bas}}^{q} + \delta^{q} \cdot \Delta t + C_{\text{sca}}^{q}(s) \right). \tag{14}$$

Therefore, we get the total resource utilization by using the following equation:

$$U^{q} = \frac{g_{\text{total}}}{g_{\text{all}}}, \tag{15}$$

where $q \in \{\text{cpu}, \text{bw}\}$. We can get bandwidth utilization and CPU utilization, respectively, by (15). From the analysis above, we get the final optimization cost as follows based on the control policy π and scaling operation:

$$g_{\text{all}} + \sum_{j=1}^{N} \sum_{s=1}^{T_j} f_j^{\pi}(s) = \sum_{j=1}^{N} \sum_{s=1}^{T_j} \left(C_{\text{bas}}^{q} + \delta^{q} \cdot \Delta t + C_{\text{sca}}^{q}(s) + f_j^{\pi}(s) \right), \tag{16}$$

where $f_j^{\pi}(s)$ is the migration cost of VM_j by the control policy π at the $s-$th timeslot.

5.1. System Configuration. We simulate two physical nodes, and each node has enough resources. VM configuration is shown in Table 1 [9, 31]. Due to the expensive CPU, we use the quantity of CPU to limit the number of VMs. The workload dataset in this paper is from the Laboratory for Web Algorithmics (LAW) (the dataset is named "eu-2015.urls.gz"; see http://law.di.unimi.it/webdata/eu-2015/ for more information). In addition, we use the Poisson process to simulate the arrival process of service requests [32, 33]. To test the performance of frameworks, we try to stabilize the arrival rate. In this paper, we set $\lambda = 8$. In the following, we will set λ from 1 to 10 for testing the robustness of SRAF.

5.2. Performance Analysis. In this section, we compare our framework (SRAF) with the framework (Symbiosis) which is proposed in [9]. We also add the Bejo algorithm [6] into the Symbiosis. In addition, we add the deadline factor into the experiments for clearly showing the difference between SRAF and Symbiosis.

In Figure 4, we make the comparison of SRAF and Symbiosis at different deadlines. All the tasks will be serviced in each framework, and they have three statuses. **Success** means that the task is finished smoothly in its owner VM. **Failed** means that the task is discarded when its service time exceeds the deadline. **Scale** means that the task needs extra resource from the resources pool for its working. In Figure 5, the results represent the utilization of bandwidth and CPU in Symbiosis and SRAF at different deadlines. In Figure 4, with the growing deadline, many failed and scale tasks become success tasks and wait for processing. The bandwidth and CPU will have more idle time. As a result, the resource utilizations are decreased as shown in Figure 5.

In Figure 4, when deadline is 20 ns and 30 ns, there are still some failed tasks in SRAF, while Symbiosis has none. When the deadline is 40 ns and 50 ns, all tasks in SRAF are success tasks, but there are also some scale tasks in Symbiosis. Therefore, SRAF has a higher absorption rate of requests and higher efficiency than Symbiosis. What caused the status above has two sides. For one thing, the more training the SCA has, the better classification results it has. The better training of SCA can make a better combination of file-focus tasks and video-focus tasks for reducing the waiting time by the sharing mode. Immediately following the operation of SCA, SRAF will make a full use of resources. As a result, SRAF has more time for working and many failed and scale tasks will become success tasks. For another thing, the sharing mode may prolong the execution time of long tasks (video tasks). But this problem can be solved by the method of resetting resource sharing proportion. For example, in Figure 5, the resource utilizations of the Symbiosis are decreasing with the growing deadline. The resource utilizations of SRAF are approximated to 97%. This phenomenon means that there are some long tasks held on CPU resources with the growing deadline. As a result, the bandwidth resources are unoccupied. Finally, the utilization of bandwidth is decreasing and that of CPU is increasing. However, SRAF has a different performance. Because of SCA, the long tasks (video tasks) are executed with short tasks (file tasks) by the sharing mode. In other words, one VM can have two tasks. When the short task has been finished and the long task is still working, a new short task will come for transmitting. As a result, the CPU and bandwidth are occupied by one task. Hence, the resource utilization of CPU and bandwidth decreases, but in a small range. However, the resource utilizations are still higher than those of Symbiosis. The detailed process of the above example is shown in Figure 2. Taking a holistic look of Figure 4, the SRAF also has a shorter dropping rate and scaling rate of all the tasks. Therefore, facing the same tasks, SRAF has a higher resource utilization and task processing rate than Symbiosis.

In order to measure the performance of our control policy π in Section 4.3, we add the operation of VM migration into Symbiosis for comparison, which is named "Symbiosis + vm-mi" (SVM). Additional execution time represents the total execution time of VM migration and scaling operation. Additional cost means the total cost of all the VM migration and scaling operations. For enhancing the veracity, we make ten experiments of SRAF, Symbiosis, and SVM, respectively, for comparison at each deadline. Let us firstly make a comparison with Symbiosis and SVM in Figures 6 and 7 because SVM makes a control policy to measure the VM migration. Therefore, when VM becomes overloaded (resource utilization exceeds thresholds), the overloaded VM will firstly choose to make migration by control policy. If not, SVM will do scaling operation (shown in Algorithms 4 and 5). On the contrary, Symbiosis can only do scaling operation for these overloaded VMs. In theory,

Table 1: VM configuration.

VM type	CPU			Bandwidth		
	MIPS (MB/s)	Per cost (dollar/h)	Per scale cost (dollar/h)	Bandwidth (MB/s)	Per cost (dollar/h)	Per scale cost (dollar/h)
High	1000	0.05	0.06	100	0.005	0.006
Normal	600	0.03	0.04	60	0.003	0.004

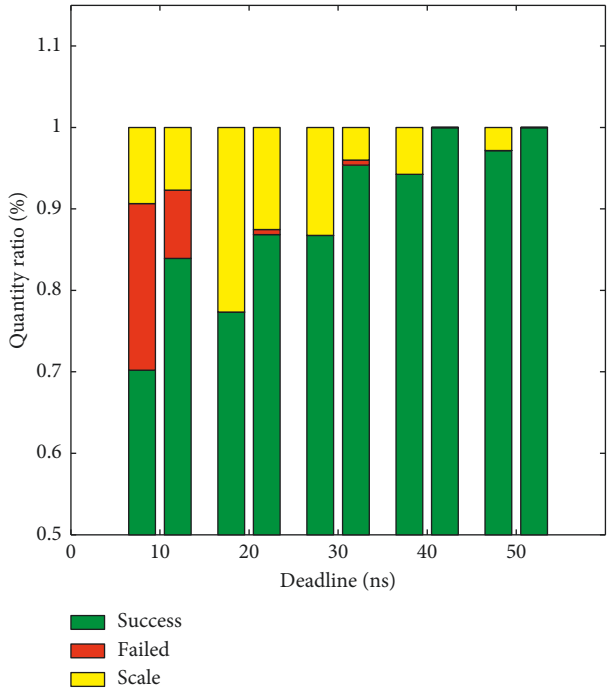


Figure 4: The quantity ratios of three statuses in all tasks.

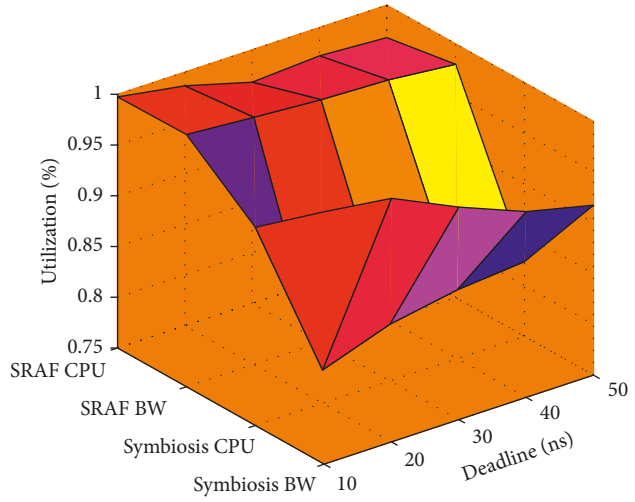


Figure 5: The utilization of bandwidth and CPU in SRAF and Symbiosis frameworks at each deadline.

VM migration can reduce the cost than scaling operation because scaling operation is easily creating frequent fluctuation. Hence, taking an overall look of Figures 6 and 7, SVM has a shorter execution time and cost than Symbiosis because the control policy can make a long-time prediction

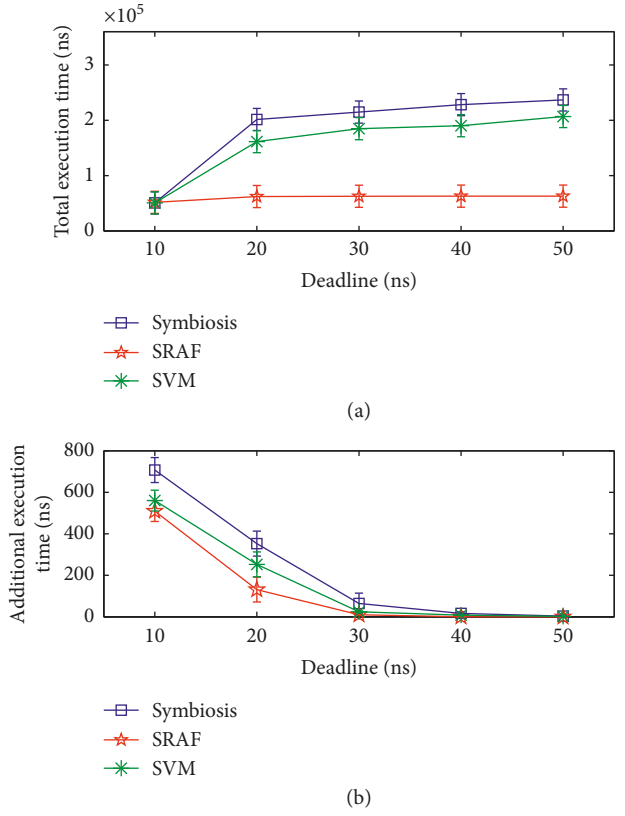


Figure 6: The total execution time (a) and additional execution time (b) of SRAF, Symbiosis, and Symbiosis + vm-mi frameworks at each deadline.

to operate VM migration so as to avoid the VM overload and resource lack. In other words, SVM uses the method of VM migration to cut down frequent scaling operation for reducing the additional cost.

From the overall perspective of Figure 6, the total execution time of Symbiosis is much more than that of SRAF. The total execution time of SRAF is almost at the level of 0.6 $\times 10^5$ ns when deadline is 50 ns. Correspondingly, the additional execution time is approximated to zero. But the total execution time of Symbiosis is almost four times higher than that of SRAF when the deadline is 50 ns. The total cost of Symbiosis in Figure 7 is also approximately five times higher than that of SRAF. The analysis above means that the service latency of SRAF is much shorter than that of Symbiosis. For example, in Figure 5, the decrease of resource utilizations on Symbiosis is sharper than that on SRAF with the growing deadline. Therefore, when more failed and scale tasks become success tasks, the total execution time of Symbiosis will continually increase. The total execution time of SRAF is changing in a small range. That is to say, facing more tasks,

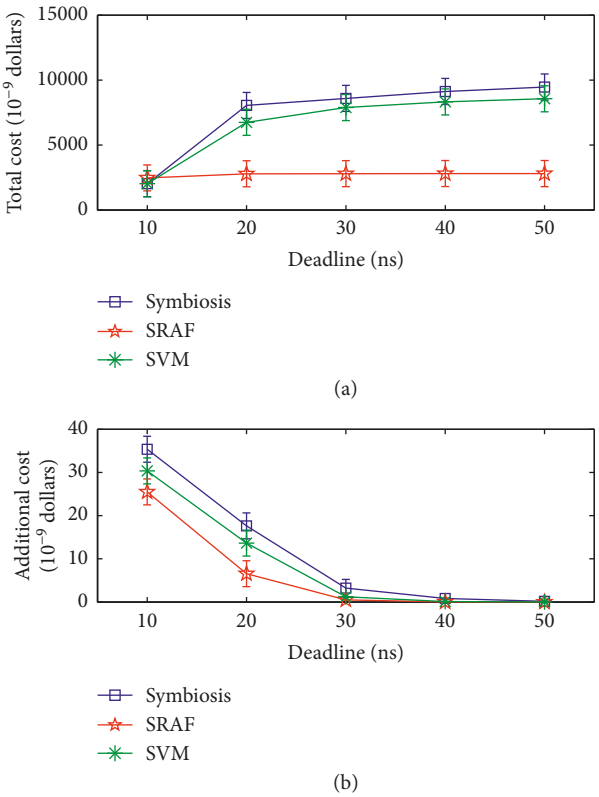


Figure 7: The total cost (a) and additional cost (b) of SRAF, Symbiosis, and Symbiosis + vm-mi frameworks at each deadline.

SRAF has a stable and better performance on reducing service latency and cost than Symbiosis.

For testing the robustness of SRAF, we set λ from 1 to 10 to simulate different situations of congestion. During experiments, we will provide enough resources. In order to avoid the additional cost made by frequent operation of changing the resources of VMs, we make the operation of resource-scaling can only maximully add twice resources of the original configuration resources on VM. We make ten experiments for each framework at different λ. Then, according to these experiments, we get the mean bandwidth utilization, mean CPU utilization, and total cost of each framework, which are shown in Figures 8–10, respectively. In Figure 8, the bandwidth utilization of SRAF, Symbiosis, and SVM is increasing with the growing value of λ. What caused the increasing phenomenon has two sides. Firstly, because the arrival rate of requests is based on the Poisson distribution, the situation of congestions becomes smooth with the growing value of λ. So, facing the more stable arrival rate, frameworks will have more time for working. As a result, all the bandwidth utilizations have an increasing trend with the growing value of λ. Secondly, if the deadline of requests exceeds the transmission time, they will become the failed tasks. In addition, we do not calculate the time and cost of failed tasks. In other words, failed tasks are the waste of bandwidth, and it is the main factor affecting the bandwidth utilization. Therefore, with the growing value of λ, the more stable arrival rate will make less failed tasks. As a result, the bandwidth utilization of frameworks is

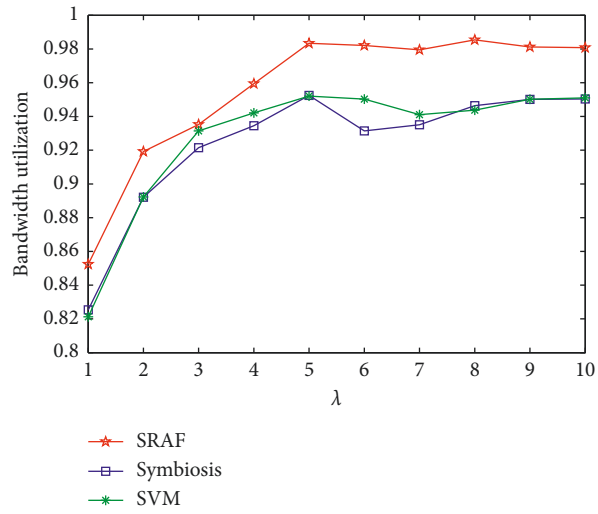


Figure 8: The bandwidth utilization of SRAF, Symbiosis, and SVM at different λ.

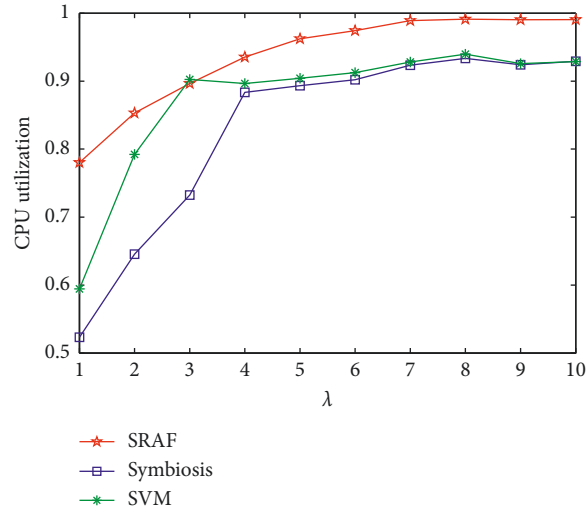


Figure 9: The corresponding CPU utilization of SRAF, Symbiosis, and SVM.

increasing. Taking a detailed look of Figure 8, it is observed that the bandwidth utilization of SRAF is higher than that of Symbiosis and SVM. Symbiosis and SVM have an approximate trend. It is because that SRAF can service two tasks simultaneously with the sharing mode (shown in Figure 2). The sharing mode can make full use of bandwidth and CPU resources than Symbiosis and SVM. Therefore, SRAF can effectively reduce the latency and enhance the utilization of free resources. The same trend of SVM and Symbiosis is that they have the same method of bandwidth transmission and do not have the sharing mode. According to these analogies above, for different arrival rates of requests, SRAF has a better performance on bandwidth utilization than Symbiosis and SVM, especially with the stable arrival rate.

Figure 9 presents the CPU utilizations of SRAF, Symbiosis, and SVM at different λ. Taking an overall look of Figures 8 and 9, CPU utilization of different frameworks is

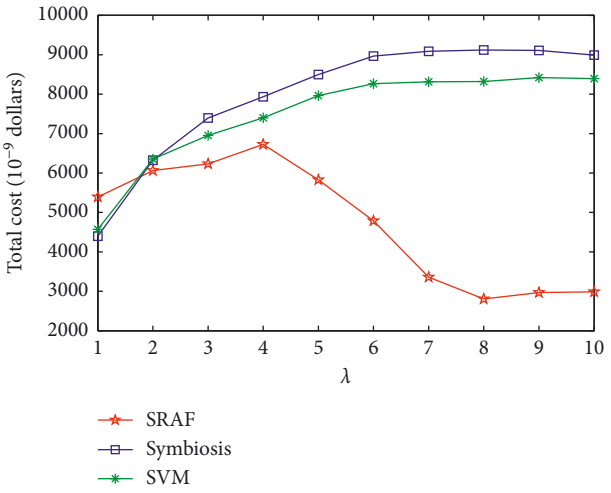


FIGURE 10: The corresponding cost of SRAF, Symbiosis, and SVM at different λ.

also increasing with the growing value of λ as the situation of bandwidth utilization. In addition, the increases of adjacent λ on CPU utilization are slightly higher than those on bandwidth utilization. It is because that the sharp congestion will cause more free time on the CPU resource than the bandwidth resource. The queueing time will exceed the deadline and cause many failed tasks. When the value of λ is growing from 1 to 4, the CPU utilizations of SVM are higher than those of Symbiosis. This phenomenon means that VM migration in SVM cuts down the quantity of scaling tasks and failed tasks. It is because that VM migration can avoid the overload of VMs and reduce the free time of failed tasks. In addition, with the growing value of λ, the arrival rate of requests is becoming stable. The stable arrival rate will reduce the frequent fluctuation of scaling operation. The operation of VM migration can also be reduced. When the value of λ is larger than 4, the CPU utilizations of SVM and Symbiosis are approximately the same. Certainly, CPU utilization of SRAF is stable and higher than that of SVM and Symbiosis because of the sharing mode and VM migration.

Figure 10 represents the total cost of SRAF, Symbiosis, and SVM at different λ. Different to the situation in Figures 8 and 9, the total cost of SVM and Symbiosis is increasing with growing λ. The total cost of SRAF is decreasing. What caused this phenomenon has two sides. Firstly, because of the higher utilizations of bandwidth and CPU in SRAF, SRAF reduces more waiting time and service time for all tasks with the sharing mode, especially for those short tasks behind the long tasks in the queue. In addition, the one-by-one service method of Symbiosis is the main factor which affects the utilizations of CPU and bandwidth. Therefore, facing the same tasks, Symbiosis will waste many resources and prolong the service time than SRAF. Secondly, VM migration can reduce frequent scaling operation. For example, the total cost of SVM is less than that of Symbiosis at each λ in Figure 10. It is because that the control policy can make a trade-off between VM migration cost and scaling operation cost. Control policy has the ability of prediction according to the history data, which can select the minimal cost of each action for reducing the additional cost. Therefore, taking a holistic look of Figures 8–10, SRAF has a better performance on stability and robustness.

6. Conclusions

In this paper, we have designed, modeled, and evaluated the SRAF, which aims to reduce the latency of service requests in mobile data networks and enhance the utilization of bandwidth and CPU resources. In SRAF, we have proposed the SCA to execute the tasks' classification. We also designed a sharing mode to realize the combination process of two tasks. Sharing mode greatly reduces the waiting time during service. In addition, we also designed an MDP to control VM migration. We use the combination method of VM migration and scaling to enhance resource utilization. Finally, we make many different experiments to show that SRAF has a good performance on resource utilization, stability, and robustness.

Data Availability

The corpus data used to support the findings of this study have been deposited in Laboratory for Web Algorithmics (http://law.di.unimi.it/webdata/eu-2015/).

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 (NSFC) under Grant nos. U1604155, 61602155, 61871430, and 61370221, in part by Henan Science and Technology Innovation Project under Grant no. 174100510010, in part by the Industry University Research Project of Henan Province under Grant no. 172107000005, and in part by the basic research projects in the University of Henan Province under Grant no. 19zx010.

References

[1] B. Yang, K. Chai, Z. Xu, K. Katsaros, and G. Pavlou, "Cost-efficient NFV-enabled mobile edge-cloud for low latency mobile applications," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 475–488, 2018.

[2] A. Mukherjee, D. De, and D. Roy, "A power and latency aware cloudlet selection strategy for multi-cloudlet environment," *IEEE Transactions Cloud Computation*, vol. 99, no. 99, p. 1, 2016.

[3] J. Li, R. Ma, H. Guan, S. David, and L. Wei, "Accurate CPU proportional share and predictable I/O responsiveness for virtual machine monitor: a case study in Xen," *IEEE Transactions Cloud Computation*, vol. 5, no. 4, pp. 604–616, 2017.

[4] B. P. R Killi and S. V. Rao, "Capacitated next controller placement in software defined networks," *IEEE Transactions on Network & Service Management*, vol. 14, no. 3, pp. 514–527, 2017.

[5] C. Birkestrand, J. Heyrman, and C. Prosser, "Virtual machine placement in a cloud computing environment based on factors including optimized processor-memory affinity," US Patent 9600321, 2017.

[6] L. Xu, J. Cao, Y. Wang, L. Yang, and J. Li, "Bejo: behavior based job classification for resource consumption prediction in the cloud," in *Proceedings of IEEE International Conference on Cloud Computing Technology and Science*, pp. 10–17, Singapore, December 2014.

[7] H. J. Kim, H. I. Kim, and J. W. Chang, "A privacy-preserving kNN classification algorithm using Yao's Garbled circuit on cloud computing," in *Proceedings of IEEE International Conference on Cloud Computing*, pp. 766–769, Honolulu, Hawaii, USA, June 2017.

[8] K. Katsalis, G. Papaioannou, N. Nikaein, and T. Leandros, "SLA-driven VM scheduling in mobile edge computing," in *Proceedings of IEEE International Conference on Cloud Computing*, pp. 750–757, San Francisco, CA, USA, June 2016.

[9] J. Jiang, S. Ma, and B. Li, "Symbiosis: network-aware task scheduling in data-parallel frameworks," in *Proceedings of the IEEE International Conference on Computer Communications*, pp. 1–9, San Francisco, CA, USA, April 2016.

[10] N. Armanfard, J. P. Reilly, and M. Komeili, "Local feature selection for data classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 6, pp. 1217–1227, 2016.

[11] H. Hajimirsadeghi and G. Mori, "Multi-instance classification by max-margin training of cardinality-based Markov networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 9, pp. 1836–1852, 2017.

[12] S. Kumar, G. Kumar, and K. Jain, "an approach to reduce turn around timeand waiting timeby the selection of round robin and shortest job first algorithm," *International Journal of Engineering and Technology*, vol. 7, no. 2, pp. 667–677, 2018.

[13] M. Kumar and S. C. Sharma, "Priority Aware Longest Job First (PA-LJF) algorithm for utilization of the resource in cloud environment," in *Proceedings of International Conference on Computing for Sustainable Global Development*, pp. 16–18, New Delhi, India, March 2016.

[14] N. T. Argon, C. Deng, and V. G. Kulkarni, "Optimal control of a single server in a finite-population queueing network," *Queueing Systems*, vol. 85, no. 1-2, pp. 149–172, 2017.

[15] B. Legros, "M/G/1 queue with event-dependent arrival rates," *Queueing Systems*, vol. 89, no. 3-4, pp. 269–301, 2018.

[16] Z. Han, H. Tan, and G. Chen, "Dynamic virtual machine management via approximate Markov decision process," in *Proceedings of IEEE International Conference on Computer Communications*, pp. 1–9, San Francisco, CA, USA, July 2016.

[17] Z. Li, "An adaptive overload threshold selection process using Markov decision processes of virtual machine in cloud data center," *Cluster Computing*, vol. 2018, pp. 1–13, 2018.

[18] S. Chen, Y. Wang, and M. Pedram, "A semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud," in *Proceedings of IEEE Global Communications Conference*, pp. 2885–2890, Atlanta, GA, USA, December 2013.

[19] T. Liu and D. Tao, "Classification with noisy labels by importance reweighting," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 38, no. 3, pp. 447–461, 2016.

[20] A. L. Jin, W. Song, and P. Wang, "Auction mechanisms toward efficient resource sharing for cloudlets in mobile cloud computing," *IEEE Transactions on Services Computing*, vol. 9, no. 6, pp. 895–909, 2016.

[21] T. X. Tran and D. Pompili, "Dynamic radio cooperation for downlink cloud-RANs with computing resource sharing," in *Proceedings of IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 118–126, Brasília, Brazil, October 2016.

[22] D. Kovachev, Y. Cao, R. Klamma et al., "Learn-as-you-go: new ways of cloud-based micro-learning for the mobile web," in *Proceedings of International Conference on Advances in Web-Based Learning*, pp. 51–61, Hong Kong, China, December 2011.

[23] C. Sing, N. Metz, and S. Dudli, "Machine learning-based classification of 38 Years of spine-related literature into 100 research topics," *Spine*, vol. 42, no. 11, pp. 863–870, 2017.

[24] J. Zhang, Y. Han, and J. Jiang, "Semi-supervised tensor learning for image classification," *Multimedia Systems*, vol. 23, no. 1, pp. 63–73, 2017.

[25] J. Chen, C. Chen, and Y. Liang, "Optimized TF-IDF algorithm with the adaptive weight of position of word," in *Proceedings of International Conference on Artificial Intelligence and Industrial Engineering*, Beijing, China, November 2016.

[26] S. I. Gass and M. C. Fu, *Bellman Optimality Equation*, Springer US, New York, NY, USA, 2013.

[27] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, Vol. 2, Nashua, NH, USA, 1995.

[28] A. Heydari, "Stability analysis of optimal adaptive control using value iteration with approximation errors," *IEEE Transactions on Automatic Control*, vol. 63, no. 9, pp. 3119–3126, 2018.

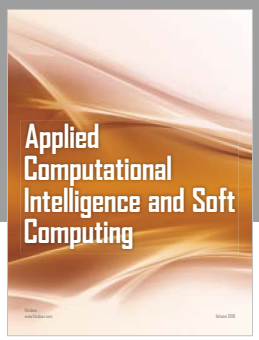
[29] B. Taylor, M. Hendrickx, and F. Glineur, "Smooth strongly convex interpolation and exact worst-case performance of first-order methods," *Mathematical Programming*, vol. 161, no. 1-2, pp. 307–345, 2017.

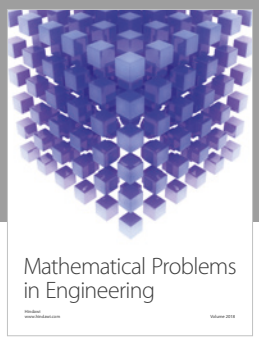
[30] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of Cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

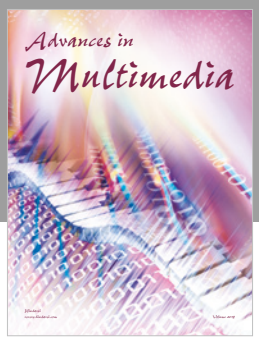
[31] Y. Zhao, N. Calheiros, G. Gange, K. Ramamohanarao, and R. Buyya, "SLA-based resource scheduling for big data analytics as a service in cloud computing environments," in *Proceedings of IEEE International Conference on Parallel Processing*, pp. 510–519, Beijing, China, September 2015.

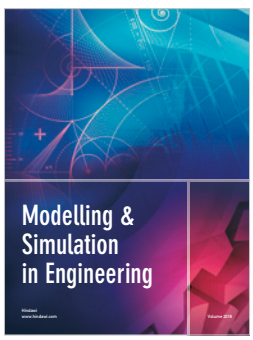
[32] Q. Xie and Y. Lu, "Priority algorithm for near-data scheduling: throughput and heavy-traffic optimality," in *Proceedings of IEEE International Conference Computer Communications*, pp. 963–972, Las Vegas, Nevada, USA, August 2015.

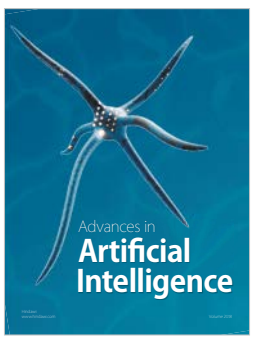
[33] S. Guo, B. Xiao, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proceedings of IEEE International Conference on Computer Communications*, pp. 1–9, San Francisco, CA, USA, July 2016.

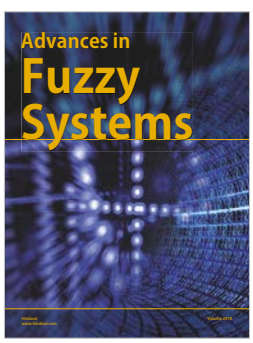


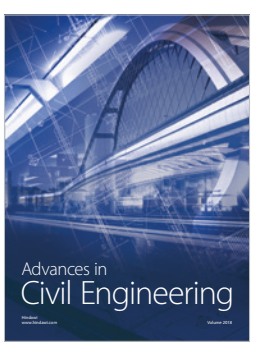


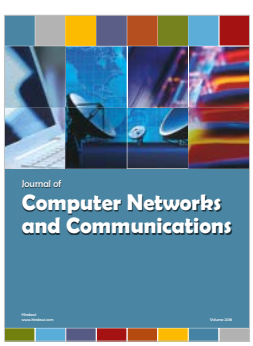


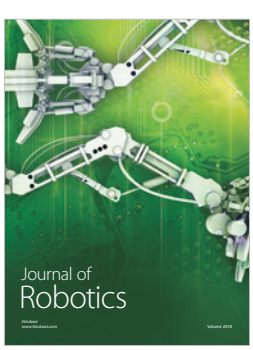


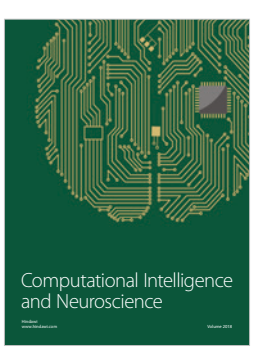












Hindawi

Submit your manuscripts at
www.hindawi.com