Hindawi
Security and Communication Networks
Volume 2021, Article ID 2729949, 17 pages
https://doi.org/10.1155/2021/2729949

Research Article

A Virtual Machine Migration Strategy Based on the Relevance of Services against Side-Channel Attacks

Ji-Ming Chen,[1,2] **Shi Chen**,[1] **Xiang Wang**,[1] **Lin Lin**,[1] **and Li Wang**[1]

[1]*School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China*
[2]*Key Laboratory of Security Technology for Industrial Cyberspace, Jiangsu University, Zhenjiang, China*

Correspondence should be addressed to Ji-Ming Chen; jmchen@ujs.edu.cn

Received 31 August 2021; Accepted 29 November 2021; Published 21 December 2021

Academic Editor: Jie Cui

With the rapid development of Internet of Things technology, a large amount of user information needs to be uploaded to the cloud server for computing and storage. Side-channel attacks steal the private information of other virtual machines by coresident virtual machines to bring huge security threats to edge computing. Virtual machine migration technology is currently the main way to defend against side-channel attacks. VM migration can effectively prevent attackers from realizing coresident virtual machines, thereby ensuring data security and privacy protection of edge computing based on the Internet of Things. This paper considers the relevance between application services and proposes a VM migration strategy based on service correlation. This strategy defines service relevance factors to quantify the degree of service relevance, build VM migration groups through service relevance factors, and effectively reduce communication overhead between servers during migration, design and implement the VM memory migration based on the post-copy method, effectively reduce the occurrence of page fault interruption, and improve the efficiency of VM migration.

1. Introduction

With the development of the Internet of Things technology, the number of Internet of Things devices is increasing rapidly, which means that a large amount of data will be generated for processing and storage. Due to the limited computing and storage capabilities of IoT devices, these data are usually uploaded to a cloud server for processing. However, the long-distance data transmission of ordinary cloud computing is difficult to meet the needs of resource-intensive and delay-sensitive IoT applications [1]. As a result, edge computing was created. Edge computing delivers ultra-low latency and high bandwidth for IoT devices to satisfy the data processing and storage requirements by putting computing and storage resources on the network edge near the IoT devices [2, 3]. The massive data generated by the Internet of Things devices in the edge computing-based IoT will contain a large amount of user identity information, location information, and sensitive information, so leakage of user information will inevitably pose significant security risks to users [4].

In edge computing, the most common method of information leakage is through side-channel attack. Malicious users use the underlying shared resources of the server to build a side channel, bypassing the logical isolation provided by the virtualized environment, and stealing the private information of other coresident virtual machines [5]. Rather than focusing on the mapping between plaintext and ciphertext, side-channel attacks obtain keys by analyzing nonfunctional behaviors and encryption or decryption operations [6, 7]. As a result, the commonly used strong encryption schemes cannot avoid exposing this physical information, posing a significant threat to the data security and privacy protection of edge computing-based IoT. Therefore, defense against side-channel attacks is crucial for edge computing. At the moment, virtual machine migration technology is frequently used to defend against side-channel attacks, which is the primary method of defense [8, 9]. Compared with traditional countermeasures, the VM migration method is general and immediately deployable [10]. By dynamically migrating virtual machines, the time for

coexistence between virtual machines is reduced, thereby reducing the amount of information that an attacker steals from the target virtual machine, so that the attacker cannot successfully obtain the target's information. Virtual machine migration can effectively prevent attackers from realizing coresident virtual machines, thereby ensuring data security and privacy protection of edge computing-based IoT [11].

As the types of IoT services increase, and their functions become more complex, a service request often requires a combination of services on multiple virtual machines. Therefore, if the association between services is not considered in the process of virtual machine migration, it will bring a large amount of communication overhead between servers and increase network energy consumption to the cloud data center. Therefore, this paper designs a virtual machine migration strategy based on service relevance, defines server relevance factors to quantify the degree of relevance between services, and migrates more relevant virtual machines to the same target server according to the degree of relevance between services. A post-copy method based on service priority is also designed to minimize the total migration time, and page fault rate. The experiment proved that the virtual machine migration strategy based on service relevance minimizes the communication overhead between servers while achieving fast and efficient VM migration.

2. Related Work

Virtual machine migration technology mainly involves server resource monitoring, load forecasting, virtual machine placement methods, and memory migration execution. In the process of VM live migration, firstly select the VM to migrate and the target server through the virtual machine placement method in the migration scheduling phase, and then complete the VM migration through the memory migration method in the migration execution phase. At present, there are many research studies on VM migration technology, which mainly focus on VM placement method and VM memory migration method.

2.1. VM Placement Methods. Virtual machine placement methods are mainly divided into migration time selection, virtual machine selection, and target server selection.

For the migration time selection problem, the prediction method is generally used to predict the resource utilization rate of the server in the future. For example, Melhem et al. [12] proposed a host load detection algorithm to determine overload or light load and migrate VMs to achieve server consolidation or load balancing.

For the VM selection and target server selection problem, Sotiriadis et al. [13] proposed an adaptive VM scheduling algorithm. The algorithm selects the VM to migrate by analyzing real-time resource monitoring data and migrates the VM to the server with the highest server evaluation value. In order to reduce the communication overhead caused by migration, Liu et al. [14] proposed a correlation-based VM migration algorithm to quantify the relationship between the resource requirements of VMs and time-varying resources, build the VM group with the highest relevance as the migration unit, and select the server with the least relevance to the virtual group as the target server. Rajabzadeh and Haghighat [15] selected the VM with the highest CPU utilization without sacrificing SLA and used the Markov chain model to select the target server. In order to reduce the resource consumption caused by migration, Xu and Fortes [16] considered the total resource waste, power consumption, and cooling cost of the VM when it was running. However, the relationship between VMs and the cost of live migration is not fully considered. In literature [17], Verma et al. considered energy consumption and migration costs, but their research showed that it is difficult to estimate the exact power consumption of the server. A novel VM placement algorithm [18] designed a new technology called resource usage factor, which can be used to quantify server resource usage and place VMs on suitable physical machines to improve the resource utilization of physical machines. In addition, Kanniga Devi et al. [19, 20] also did research on optimizing placement methods. They used cluster intelligent algorithms to optimize the selection of VMs or target servers in VM migration strategies, reducing energy consumption in cloud computing and improving Resource utilization rate.

2.2. VM Memory Migration Methods. The research of memory migration methods mainly focuses on the pre-copy method and the post-copy method [21].

Mandal et al. [22] designed an algorithm to find the appropriate bandwidth and the number of prereplication iterations. It develops a model to measure network resource consumption, migration time, and downtime and determine the appropriate migration bandwidth and number of prereplication iterations to improve performance. In order to reduce the total migration data, the paper [23] proposed a method to optimize the pre-copy method, which can reduce unnecessary memory page transfer, and the feature-based compression (CBC) algorithm reduces the total migration time and downtime. In order to ensure that the memory migration is completed within a specific time, Zhang et al. [24] proposed a novel transmission control mechanism to ensure the bandwidth of the calculation and theoretically analyzed the bandwidth demand to ensure the total migration time and downtime. Literature [25] conducts a comprehensive empirical study on the pre-copy method to provide suggestions for optimal selection. To optimize post-copy method, Su et al. [26] improved the subsequent copy method by eliminating unnecessary remote page errors.

Sun et al. [27] proposed an improved serial migration strategy, which is based on the post-copy method and supports multivirtual machine migration. Lei et al. [28] proposed a hybrid copy method, which combines the pre-copy method with the subsequent copy method to make up for the defects of the pre-copy method and the post-copy method, but this method still has page fault. Deshpande et al. [29] designed an eviction-aggregation VM live migration method. It speeds up the eviction time of the VM migration

process by setting the cache area and restores the overloaded server to the normal state in the fastest time.

At present, when selecting multiple VMs for migration, the research on VM selection seldom considers the communication consumption between the VMs. If closely related VMs are migrated to different servers, it will bring more data transmission overhead between servers in the calculation process. This paper defines service relevance factor to quantify the degree of relevance between services and proposes a VM selection strategy based on service relevance. It combines closely connected VMs into a VM migration group for migration to reduce energy consumption and communication overhead. For memory migration methods, this paper chooses to improve the post-copy method and proposes a post-copy memory migration method based on service priority (PBSCP). The method sets the initial priority of the service based on service relevance and updates the service priority according to the page fault situation to reduce the occurrence of page fault interruption. This method also reduces the migration time and the migration data volume by adding temporary storage devices and reduces the occurrence of page faults.

3. The Design of Virtual Machine Migration Strategy

3.1. Service Relevance Factor. Figure 1 describes the communication relationship between devices in the Edge Computing. Users and IoT devices upload the collected data to the edge server through the network for processing and storage, and then the VMs on the server provide services to users and IoT devices, so the edge server will generate interserver communication overhead due to the cooperation between services. The dynamic nature of multiple service relationships affects the communication overhead between servers in the process of VM migration. Before designing VM selection and memory migration strategies based on service relevance, we need to quantify the degree of association between services, which determines the value of the service relevance factor.

This paper defines the service relevance factor as the degree of relevance between application service programs on each VM. Before measuring the degree of association, the interaction rate between application services and the communication overhead of each interaction need to be considered. The service set is defined as $I = \{I_1, I_2 \dots I_k \dots I_n\}$, and there are different interaction rates and communication overheads between services.

Assume that service I_i on virtual machine V_k and service I_j on virtual machine V_l have an interaction relationship when providing services to users. The interaction rate factor is defined as $IR_f^{ij}(V_{kl})$, which represents the number of interactions required to process service requests per second. According to the interaction factor, define the server overhead $SC(I_i, I_j)$ consumed by two related services as

$$SC\left(I_i, I_j\right) = IR_f^{ij}\left(V_{kl}\right) \cdot \Delta c_{ij}, \tag{1}$$

where Δc_{ij} represents the cost of processing each interactive task.

Since a VM can contain multiple services, the relationship between services is also different. If there is no interaction between services, the service overhead is 0. Therefore, the calculation formula of service relevance factor can be expressed as

$$SC\left(I_i, I_j\right) = \begin{cases} IR_f^{ij}\left(V_{kl}\right) \cdot \Delta c_{ij}, & I_i \text{ is related to } I_j, \\ 0, & \text{others.} \end{cases} \tag{2}$$

3.2. VM Selection Strategy Based on Service Relevance. A virtual machine migration strategy is based on the relevance of services against side-channel attacks, a VM group with a higher association is built through the service relevance factor, and the VM migration group is used as a whole for migration. From the perspective of service relevance, the VM migration group is a closely connected "area" on the server. Migrating this whole to the target server can improve task execution efficiency and reduce network communication overhead between servers.

During the construction and expansion of the VM migration group, the VM with the highest relevance within the group should be found. Therefore, in the process of building a VM migration group, it is necessary to compare the service relevance C^{GI} between a certain VM and the migration group, and the service relevance C^{GO} between this VM and outside the group.

This paper defines $CF(V_i, G)$ to indicate the association between VM V_i and VM migration group G. The calculation of $CF(V_i, G)$ can be expressed as

$$CF\left(V_i, G\right) = C_{V_i,G}^{GI} - C_{V_i,G}^{GO}. \tag{3}$$

According to the calculation formula of service relevance factor (2), it can be seen that the server cost between any two VMs is shown in the following formula:

$$\begin{aligned} &C^L\left(V_i, V_k\right) \\ &= \sum_{I_x \in V_i} \sum_{I_y \in V_k} SC\left(I_x, I_y\right) \\ &= \sum_{I_x \in V_i} \sum_{I_y \in V_k} IR_f^{xy}\left(V_{ik}\right) \cdot \Delta c_{xy}. \end{aligned} \tag{4}$$

Formula (4) can be further expressed as

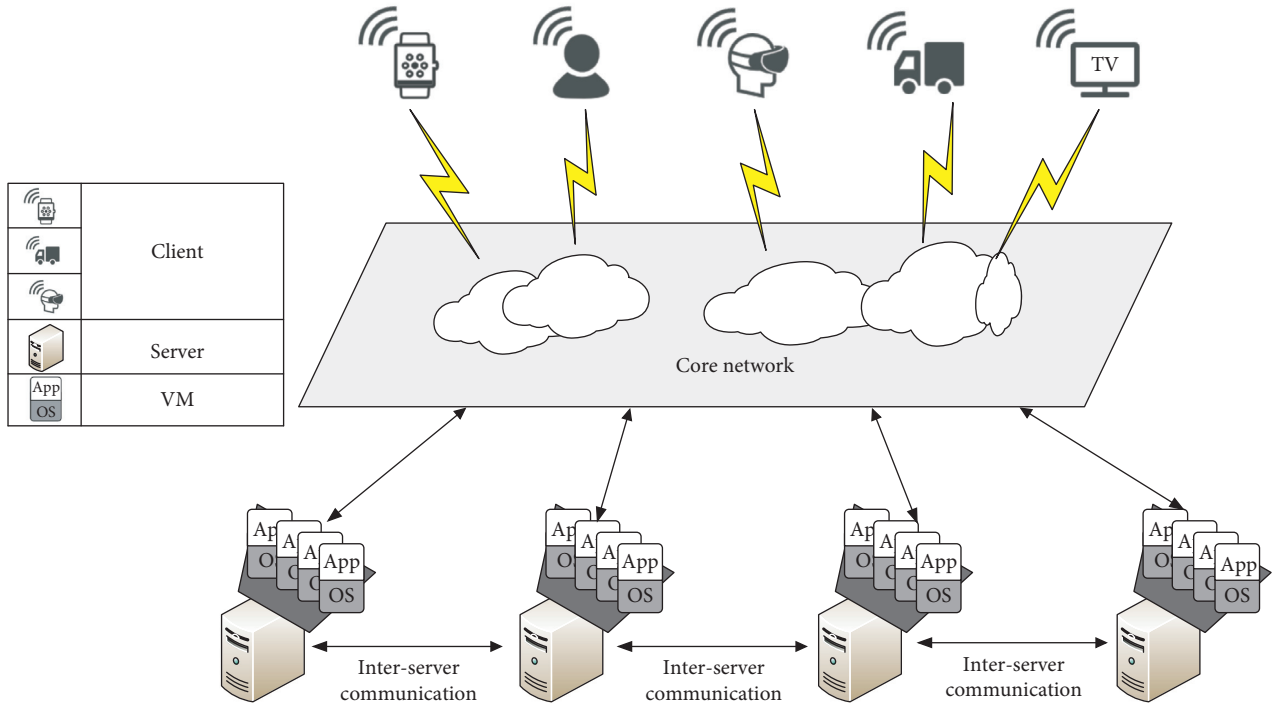

Figure 1: Communication relationship between devices in the edge computing-based IoT.

$$\begin{aligned} &CF(V_i, G) \\ &= \sum_{V_k \in G}^{C^L} (V_i, V_k) - \sum_{V_l \notin G}^{C^L} (V_i, V_l) \\ &= \sum_{V_k \in G} \sum_{I_x \in V_k} \sum_{I_y \in V_i} IR_f^{xy}(V_{ki}) \cdot \Delta c_{xy} - \sum_{V_l \notin G} \sum_{I_z \in V_l} \sum_{I_y \in V_i} IR_f^{zy}(V_{li}) \cdot \Delta c_{zy}. \end{aligned} \tag{5}$$

Set the server overload threshold $S_{\text{threshold}}$. If the service load exceeds the threshold, it will be regarded as an overload state, in order to facilitate the measurement of the load condition that the server can accept when it is not overloaded. The load capacity of the server S_i is defined as W_i^{accept}.

In this paper, the VM selection strategy is the construction process of the VM migration group. When determining the amount of migration, find the VM with the largest load on the original server as the initial migration group, and then expand the initial migration group to include more load.

During the expansion process, the VM with the closest relationship to the group must be selected each time, and the load of this VM and the load of the migration group must be calculated whether the total load reaches the migration data volume. If the required volume of migration data is not reached, the VM is added to the group; otherwise, the next closest VM is selected, and the process is repeated until the load of the migration group meets the requirements. The specific strategy is shown in Figure 2.

Assuming that the load to be migrated for original server S_i is W, the acceptable load difference threshold is W_{bound}, the VM on S_i is denoted as $\{V_1, V_2 \ldots V_m\}$, the VM migration group constructed is $G = \{V_1, V_2 \ldots V_k\}$, and the construction algorithm of the VM migration group is as shown in Algorithm 1.

3.3. Memory Migration Strategy Based on Service Priority. The VM migration group constructed based on service relevance has the characteristics of large data volume and many service requests. This paper combines service relevance and designs a post-copy memory migration method (PCBSP) based on service priority. The method uses service priority to determine the corresponding memory page push priority and adds temporary storage devices (TD) during the migration process to make the migration time minimize and reduce the page fault rate.

3.3.1. Overall Process of PCBSP. In order to minimize the VM migration time and reduce the page fault rate, PCBSP adds a memory page push algorithm based on service priority and temporary storage devices on the basis of the post-copy method. The basic process of PCBSP is shown in Figure 3.

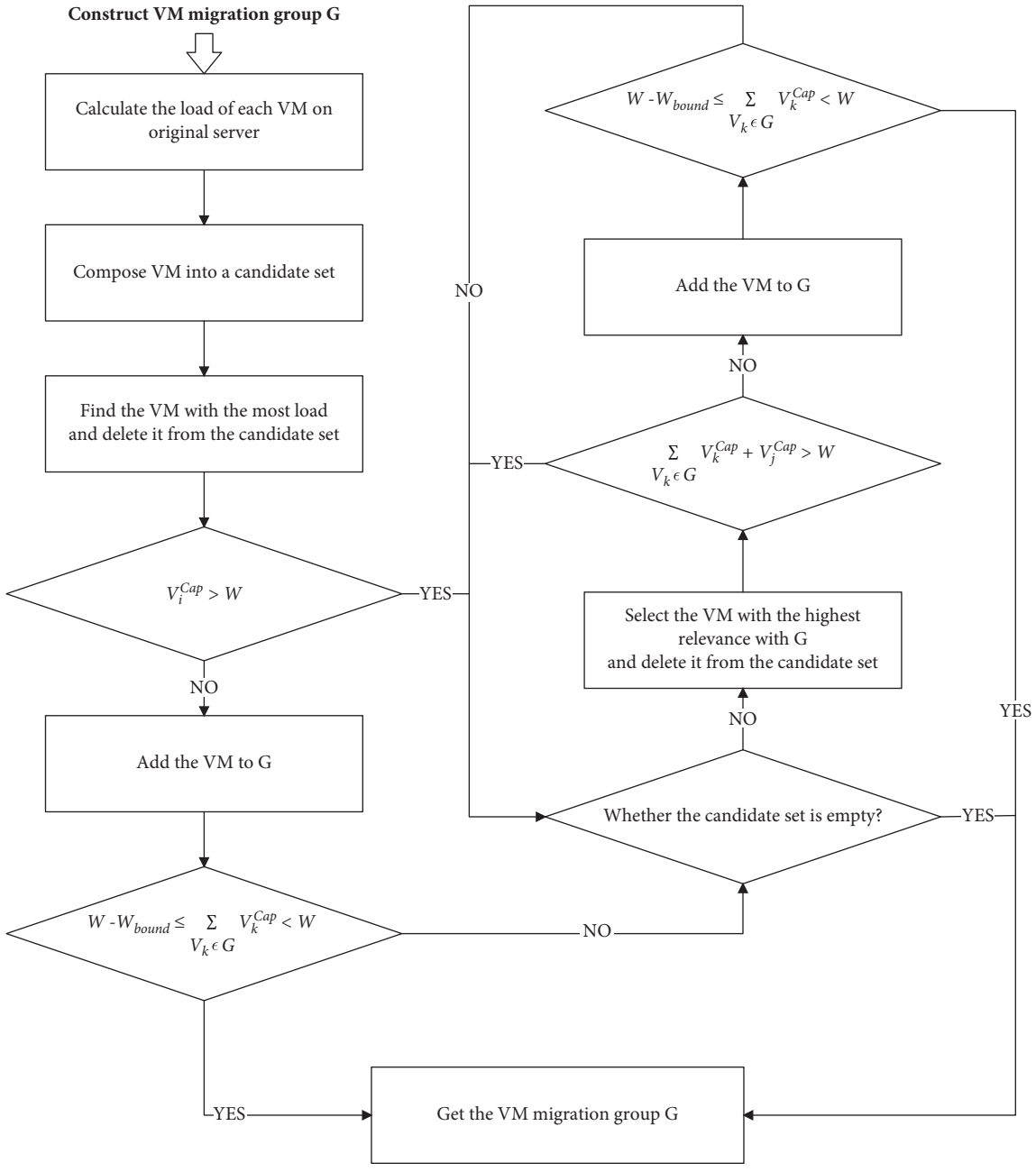

Figure 2: Process of building a VM migration group.

PCBSP mainly includes three stages. The first phase needs to select the appropriate spare memory construction in the physical server. The second phase is the VM migration phase, which includes migrating memory pages to TD and sending memory pages directly to the target server.

The third phase is the VM migration phase, which includes the migration of memory pages from TD and the direct acquisition of required memory pages from the original server. The migration of memory pages is collectively called the memory page push.

3.3.2. Priority Calculation. Since multiple services can be deployed on a VM, that is, the VM migration group contains several services, and each service provides a service when the user requests it, there may be dependent or dependent relationships among multiple services. In this paper, a memory page push algorithm (PBSP) based on service priority is designed to actively push memory pages to reduce the occurrence of page fault in the target server.

When the VM is resumed on the target server, the initial priority of the service and related pages is calculated according to the service request rate and the dependency relationship between the services.

Definition 1 (service rate). For a certain service I_j in the VM migration group G, the number of requests received from users per unit time is taken as the service rate, which is recorded as $SR(I_j)$.

Input: server load information
Output: VM migration group G
Method:
(1) Analyze the server load information to get the original server as S_i;
(2) Get the set of virtual machines on S_i as AllVmList, set to $\{V_c^*\}$;
(3) Calculate the VM load on S_i, mark it as $\{V_i^{\text{Cap}}\}$, and sort in descending order of load;
(4) Initialize the VM migration group G;
(5) **while**($\{V_c^*\}$! = null) **do**
(6) find the VM with max(V_i^{Cap}) and set it as V_i;
(7) $\{V_c^*\} = \{V_c^*\}/V_i$;
(8) **if** $(V_i^{\text{Cap}} < W)$ **then**
(9) add V_i to G;
(10) **if** $(W - W_{\text{bound}} \le V_i^{\text{Cap}} < W)$ then
(11) **return** VM migration group G;
(12) **else**
(13) **break;**
(14) **end if**
(15) **end if**
(16) **end while**
(17) **while**($\{V_c^*\}$! = null) **then**
(18) select $\{V_c^*\}$ in max$(CF(V_i, G))$, that is, select the VM with the most service relevance and set it to V_j;
(19) $\{V_c^*\} = \{V_c^*\}/V_j$;
(20) **if**$(\sum_{V_k \in G} V_k^{Cap} + V_j^{Cap} < W)$ **then**
(21) add V_i to G;
(22) **if** $(W - W_{\text{bound}} \le \sum_{V_k \in G} V_k^{Cap} + V_j^{Cap} < W)$ **then**
(23) **break;**
(24) else
(25) continue;
(26) end if
(27) end if
(28) end while
(29) **return** VM migration group G;

Algorithm 1: Build VM migration group.

Figure 4 is an example graph of an associated service group. Services are represented by dots in the graph, and service dependencies are represented by directed edges in the graph.

Definition 2. According to the dependency between services in the associated service group, define the set of service nodes, which is a dependence of service I_j as reply(I_j) and the set of service nodes, which is dependent on I_j as depended(I_j).

Set the default priority of Service I_j as

$$DR(I_j) = \alpha \frac{SR(I_j)}{\sum_{I_i \in G} SR(I_i)}, \tag{6}$$

where α represents a parameter that can simplify $DR(I_j)$ to an integer.

Each service performs related functions and needs to load the corresponding program into the memory.

Through this loading mechanism, the corresponding page fault rate can be inferred according to the probability of service access. Therefore, when calculating the preset priority, the service priority and the related memory page priority can be equivalent. Thus, the default priority of memory pages related to I_j can be obtained:

$$DR(\text{pages}(I_j)) = DR(I_j). \tag{7}$$

For a service, which is being dependent on, the greater its out-degree, the greater the |depended(I_j)|, and the priority push of its related pages can prevent page faults due to the lack of dependent services when users request the service, which is being dependent on, which can effectively reduce the page fault rate. This paper comprehensively considers the in-degree and out-degree of the service and defines the initial priority calculation method for the service and its related pages:

$$\begin{aligned} PR(\text{pages}(I_j)) &= PR(I_j) \\ &= DR(I_j) \cdot \left(\beta_1 \left|\text{depended}(I_j)\right| + \beta_2 \left|\text{reply}(I_j)\right|\right) \\ &= \alpha \frac{SR(I_j)}{\sum_{I_i \in G} SR(I_i)} \cdot \left(\beta_1 \left|\text{depended}(I_j)\right| + \beta_2 \left|\text{reply}(I_j)\right|\right). \end{aligned} \tag{8}$$

For page fault service I_k, when the VM resumes running on the target server, first check whether there are any memory pages for service I_k, and if so, push the memory pages according to the initial priority calculated by (8).

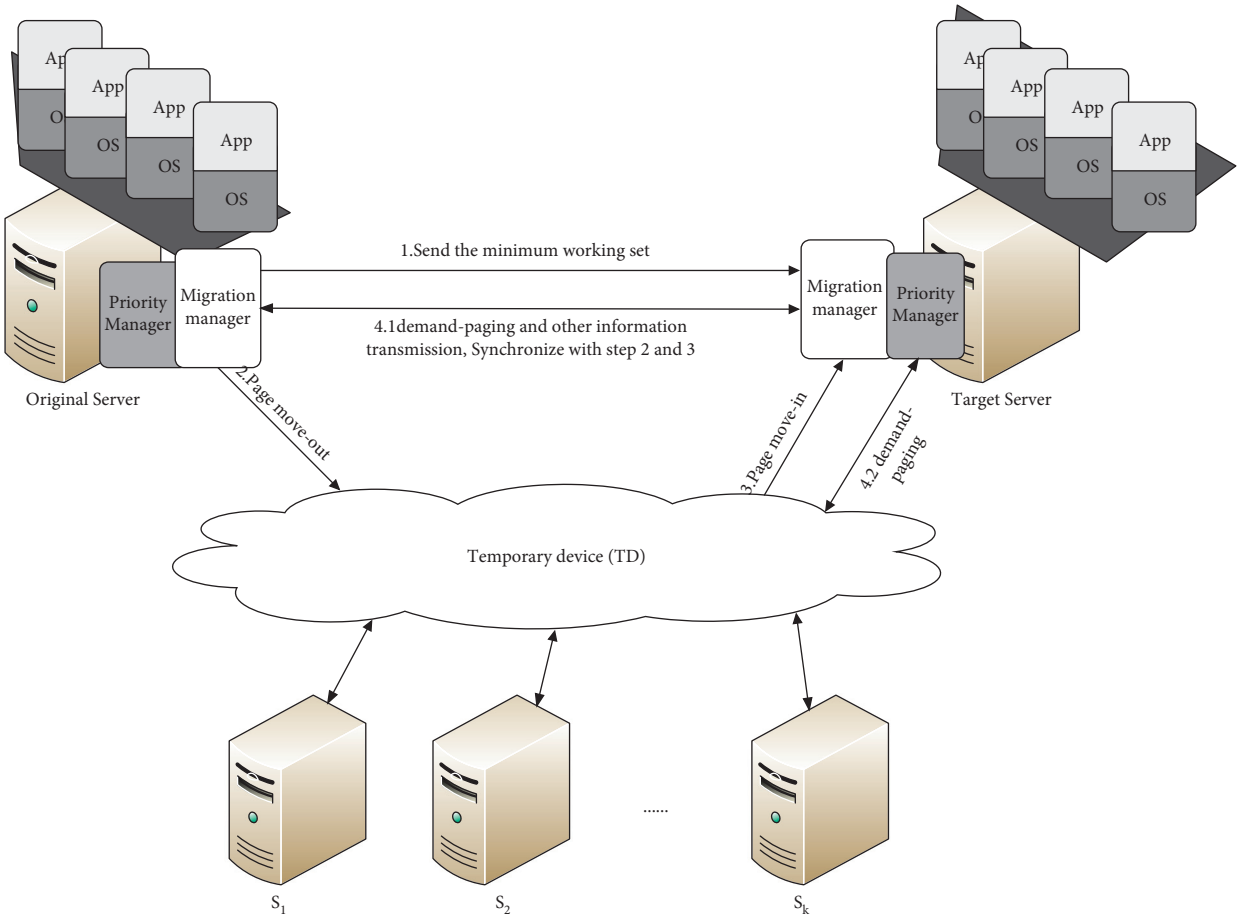

Figure 3: Overall process of PCBSP.

The priority of memory page push is mainly determined by |reply(I_j)|, |depended(I_j)|, and $SR(I_j)$. Since |reply(I_j)| and |depended(I_j)| are fixed amounts, the service rate $SR(I_j)$ is an average value for a period of time, which is not related to the interaction sequence. If the interaction rate is low, but the access is earlier, according to the priority, it may not be pushed first, and page faults will eventually occur. Therefore, the priority needs to be dynamically adjusted according to the real-time situation of service access during the memory migration process.

In order to dynamically adjust the priority, this paper introduces the following parameters: V_0, T, push_tab1 and push_tab2, where V_0 represents the change value of the priority related to page faults when the original server receives a page fault request, T represents the priority update cycle, and push_tab1 represents the memory page priority array that has not yet been migrated to T D, and push_tab2 represents the memory page priority array on T D.

The push_tab1 and push_tab2 priority update methods are the same, and the two arrays are collectively referred to as push_tab.

Set the index of memory page in the priority array push_tab as push_index, service I_j of page, and if there is no page fault request related to service I_j within T, the priority is reduced as

$$\begin{aligned}&\text{push\_tab}[\text{page\_index}]\\&= \frac{\text{push\_tab}[\text{page\_index}] + \text{PR}\left(\text{pages}\left(\text{I}_\text{j}\right)\right)}{2}.\end{aligned} \tag{9}$$

If a page fault request for a service related to page occurs within T, the page faulted service is I_k. If this service is a dependence of the service in page, the priority is increased to

$$\begin{aligned}&\text{push\_tab}[\text{page\_index}]\\&= \text{push\_tab}[\text{page\_index}] + \frac{\text{V}_0}{\left|\text{depended}\left(\text{I}_\text{k}\right)\right|}.\end{aligned} \tag{10}$$

If this service is dependent on the service in page, the priority is increased to

$$\begin{aligned}&\text{push\_tab}[\text{page\_index}]\\&= \text{push\_tab}[\text{page\_index}] + \text{V}_0.\end{aligned} \tag{11}$$

Because once a page fault occurs, as a dependence service, there is a high probability of a page fault.

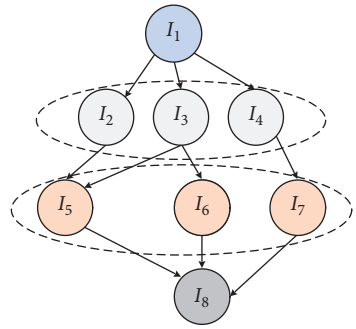

Figure 4: Example of associated service group.

3.3.3. Implementation of PBSP. When a large number of page faults occur in the restored VM, it directly requests memory pages from the original server. At this time, the page fault transmission is directly transmitted from the original server to the target server in descending order of the initial priority. The remaining memory pages in push_tab adjust their priority dynamically and actively push memory pages to $T\ D$ in descending order of priority.

The memory page push algorithm (PBSP) based on service priority is given in Algorithm 2.

4. Implementation

This section mainly introduces the overall process of the VM migration strategy based on service relevance and introduces the VM selection strategy based on service relevance in the migration scheduling phase and the specific implementation of the memory migration method based on service priority in the memory migration phase.

4.1. Overall Process. The overall process of VM live migration is divided into two phases, namely, the migration scheduling phase and the migration execution phase. In this paper, the VM selection strategy based on service relevance is applied to the migration scheduling module, and the memory migration method based on service priority is applied to the migration execution module. The overall framework of VM live migration is shown in Figure 5.

The main process is divided into two phases: the migration scheduling phase and the migration execution phase. The migration scheduling phase mainly involves VM selection and the target server selection, and the migration execution phase mainly involves memory migration.

4.2. Implementation of Migration Scheduling Phase. The migration scheduling phase is mainly divided into two parts. Firstly, the load that needs to be migrated is calculated according to the original server's own load information and builds the VM migration group. Then, the target server is selected to place the VM migration group according to the load information of other servers provided by the VM migration management center. The selection of VMs is the key to migration scheduling. This paper uses a VM selection strategy based on service relevance to construct a VM migration group. The process of the migration scheduling phase is shown in Figure 6.

Step 1: after the server management center detects the migration command, it starts the construction of the VM migration group and builds the VM migration group G according to the service relevance as the load group that needs to be migrated. Then, send a VMM request [Req] to the VMM management center.

Step 2: after the idle server receives the VMM request information, it calculates its own failure rate, load capacity, and other information.

Step 3: the idle server returns the reply message [reply] to the original server.

Step 4: if the original server is adjacent to the idle server, that is, there is a TCP link, send a reply message [reply] directly to the original server; otherwise, establish a new link first, and then send [reply] to the server.

Step 5: the original server receives [reply], selects the appropriate target server according to the load information of the idle server, and prepares to enter the migration execution phase.

4.3. Implementation of the Migration Execution Phase. The migration execution phase mainly performs VM migration. The first step is to establish a temporary storage device to calculate the service priority according to formula (8). The second step is to directly send the original server to the target server and migrate the memory page to the TD through the running state of the VM. In the third step, the target server moves out the memory page from the TD and obtains the required memory page from the original server. The second and third steps are carried out at the same time and are collectively referred to as memory page push. Figure 7 shows the process of the migration execution phase.

Next, the specific implementation process of the migration execution phase is described in detail:

Step 1: the original server obtains the resource usage of each server from the VMM manager and obtains the candidate server list $H = S_1, S_2 \ldots S_m$ of TD according to the result of the migration scheduling stage. Select some physical servers in the list as TD, which can be used by overloaded servers to quickly move out of VMs and move in memory pages to the target server.

Input: initial list push_tab
Output: push_tab
Method:
(1) Receive page faults request;
(2) Send pages to target server;
(3) push_tab.delete (pages);
(4) Service I_1 = getService (page);//get service related to fault page.
(5) **for** each page in unMigrationPages **do**//unMigrationPages are pages unmigrated.
(6) Service I_j = getService (page);
(7) **if** (I_j is related to I_1) **then**
(8) **if** (I_j replay on I_1) **then**
(9) push_tab[page_index] = push_tab[page_index] + (V_0/|depended(I_1)|);
(10) **else**
(11) push_tab[page_index] = push_tab[page_index]+V_0;
(12) **end if**
(13) **else**
(14) push_tab[page_index] = push_tab[page_index] + PR((pages(I_j))/2);
(15) **end if**
(16) **end for**
(17) **return** push_tab;

Algorithm 2: Page push algorithm based on service priority (PBSP).

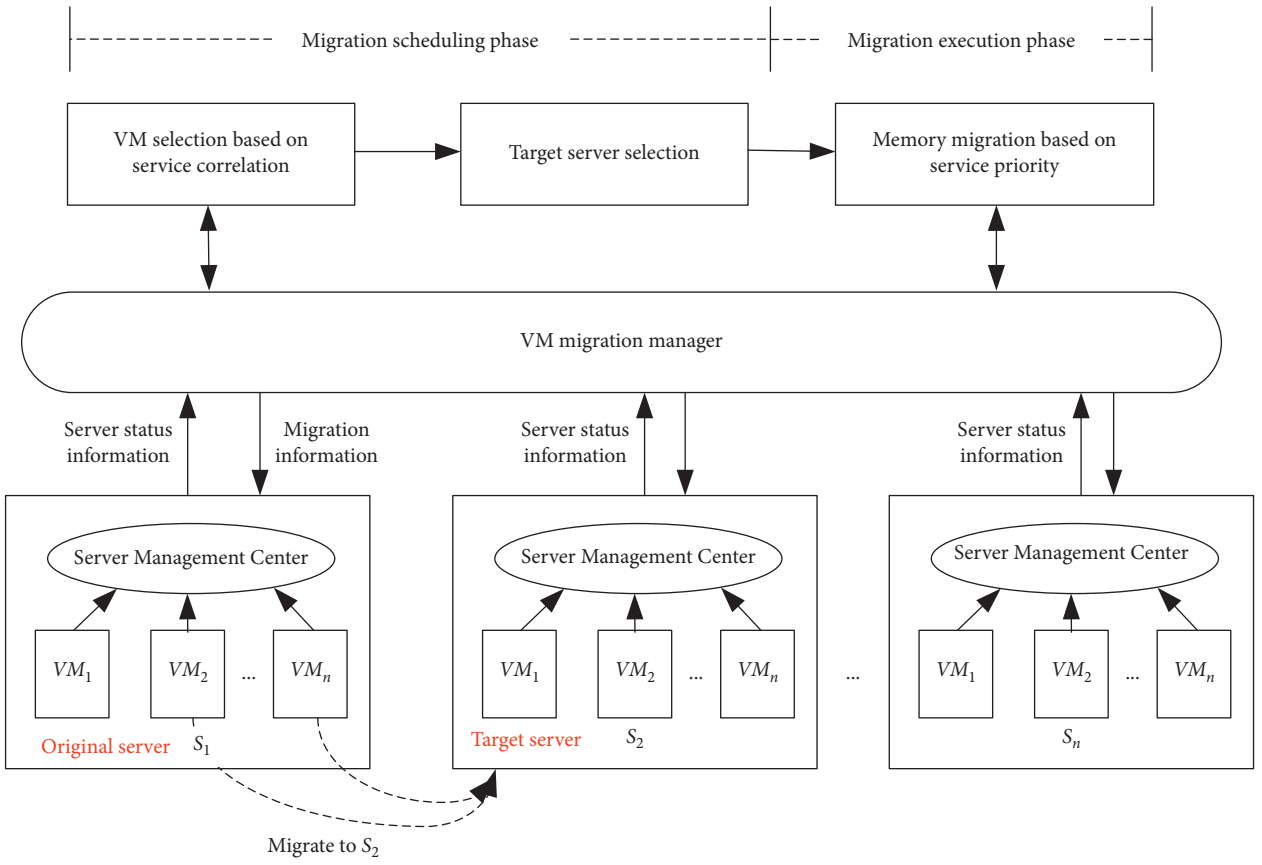

Figure 5: VM live migration framework.

Step 2: send the minimum execution conditions such as the execution status of each VM in the VM migration group directly to the target server and resume the execution of the VM on the target server.

Step 3: move the memory pages in the VM migration group into the TD in the order of service priority. If the VM migrated on the target server has a page fault interruption at this time, it directly requests the

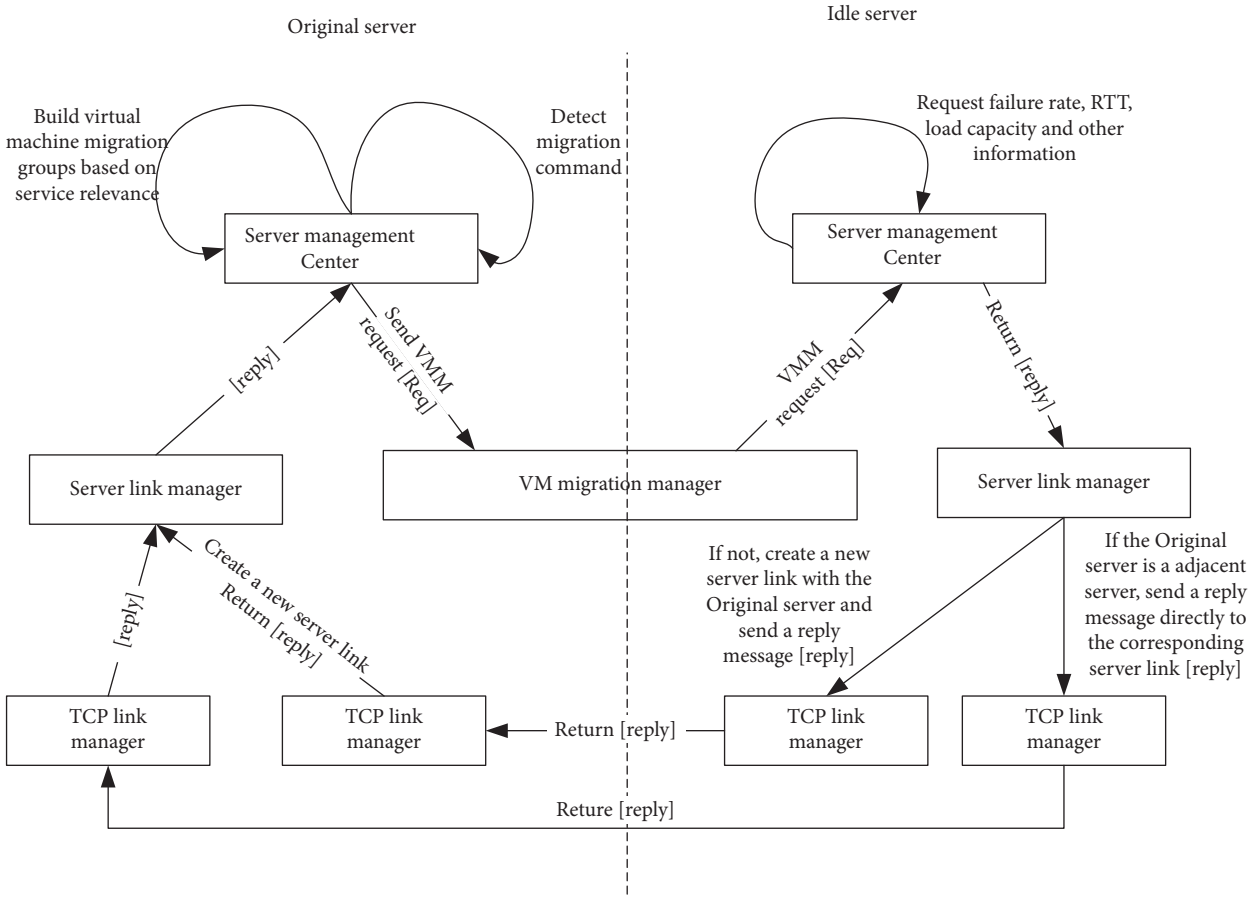

Figure 6: Migration scheduling stage.

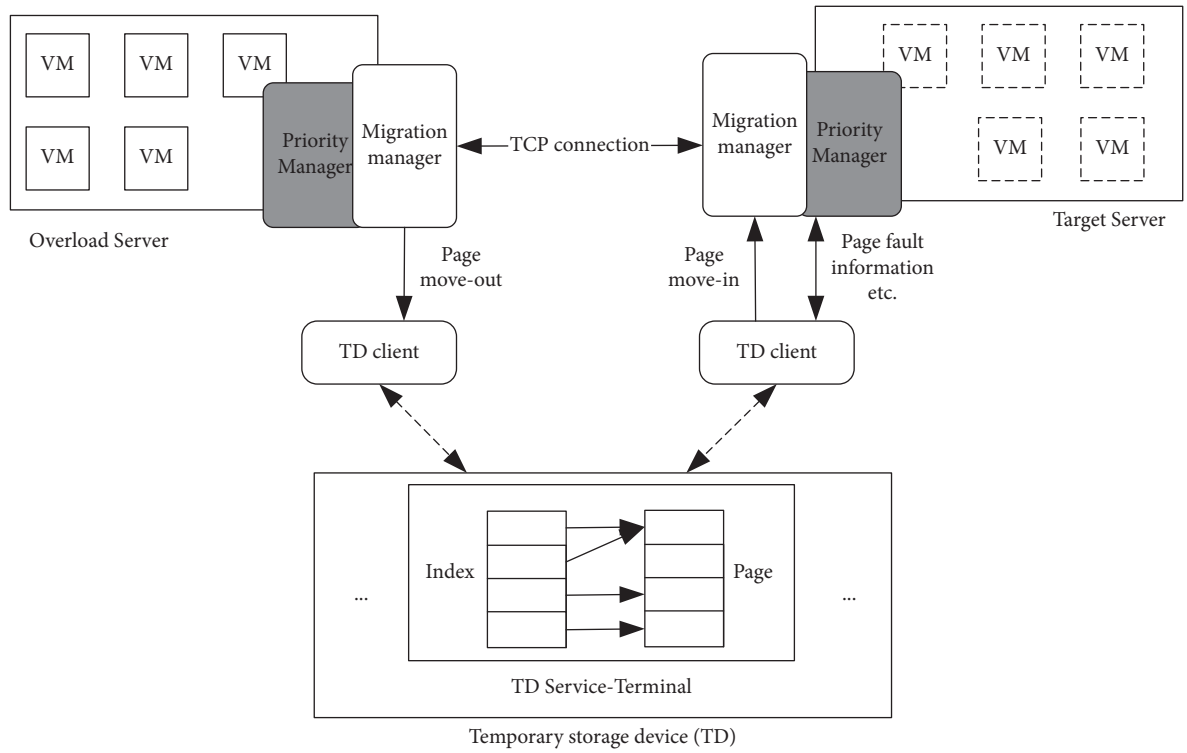

Figure 7: Migration execution phase.

memory page from the original server, and the original server sends the page fault to the target server and, at the same time, updates the order in which the VM memory page is pushed. Repeat this process until all the memory in the VM is sent to the TD.

Step 4: the target server migrates the memory pages in the TD to the server in the order of service priority. This process is synchronized with the third step. If a page fault occurs after the end of the third step, a memory page is directly requested from the TD. Repeat this process until the target server finishes receiving memory.

5. Results Analysis

5.1. Experimental Protocol. In the experiment, this paper uses a total of 6 PCs based on the actual situation, of which one is used as a VMM management node to deploy migration management strategies, and the other 5 are used as edge nodes. The configuration of each PC is shown in Table 1.

The Eucalyptus cloud computing platform is open source, easy to use, and rich in management interfaces [30]. This article uses the Eucalyptus architecture to build the above devices into a cluster system. In this experiment, the management node is set as the VM migration manager, and the VMs are run on edge nodes to simulate providing services to terminal devices. This article deploys the VM placement method and memory migration method on the system to implement the VM migration framework. The topology of the final cluster is shown in Figure 8.

All VMs in the system use KVM/QEMU 1.6.50, running Ubuntu as a suboperating system, with 2 virtual CPUs (vCPU), and each VM is configured with different memory sizes such as 512M, 1024M, 2G, etc.

In order to verify the effectiveness of the VM migration strategy based on service relevance, the experiment was divided into four groups, and the following questions were tested, respectively:

(1) Whether the strategy can reduce the communication overhead between servers?

(2) Whether the strategy can load balance quickly?

(3) Whether the memory migration method of this policy is efficient?

5.2. Analysis of Results

Experiment 1. This experiment is to verify the effectiveness of the VM placement method based on service relevance (SRVMP) and simulate the round-robin scheduling (RR) algorithm [31], the least connection (LC) algorithm [32], and the ant colony (AG) algorithm [33]. In order to compare the communication overhead of each algorithm, this paper divides each algorithm into single VM migration (single) and multiple VM migration (multiple). The experimental comparison chart is shown in Figure 9. It can be seen from Figure 9 that the communication overhead of the SRVMP algorithm is much smaller than that of the RR algorithm and the LC algorithm, and the communication overhead of multivirtual machine RR algorithm is lower than single-virtual machine RR algorithm, and multivirtual machine LC algorithm has lower communication overhead than single-virtual machine LC algorithm. This is because multivirtual machine migration algorithm migrates multiple VMs to the same server, which avoids the communication overhead caused by VMs being scattered to different servers. The SRVMP algorithm considers the service relevance between VMs during multivirtual machine migration and groups closely related VMs into a migration group. Therefore, the SRVMP algorithm designed in this paper can reduce the communication overhead between servers during migration.

Experiment 2. Experiment 2 needs to test the load balancing ability and task execution time of the system, and the combined result of the two is used as a measure of the effectiveness of load balancing.

The load balancing capability is measured by the load balance degree. The load balance degree refers to the standard deviation between the server resource utilization and the average data center resource utilization, which reflects the system load distribution. And the smaller the degree is, the more balanced the load is. The calculation formula is as follows:

$$L_{\text{factor}} = \sqrt{\frac{\sum_{i=1}^{n} \left(L_i - \overline{L}_{\text{datacenter}}\right)^2}{n}}, \tag{12}$$

where L_{factor} represents the load balance degree, n represents the total number of servers in the data center, L_i represents the resource utilization of the servers S_i, and $\overline{L}_{\text{datacenter}}$ denotes the average utilization of the data center resources.

The task execution time indicates the duration from entering the processing queue to completing the task processing. This paper mainly takes the form of randomly generating multiple tasks, such as generating the number of $10^3 - 5 * 10^3$ tasks at regular intervals and then recording the results of several time units. In the task processing time experiment, the efficiency of different algorithms is verified by different task numbers. Besides, so as to avoid accidental factors, the average is taken as the final result through 30 experiments.

The algorithms in this experiment are all based on multivirtual machine migration, and on the basis of Experiment 1, a self-heuristic algorithm ant colony (AG) algorithm is added for comparison. The results of the experiment are shown in Figure 10.

According to Figure 10(a), we can see that the SRVMP algorithm is relatively evenly distributed in the CPU load balance degree. Although it is lower than the AG algorithm, the load balancing degree is roughly distributed around 10. The reason for this phenomenon is that this paper uses the greedy strategy to chooses the server with the least migration cost as the target server, so the communication cost and other indicators are not optimal. Comparing the RR algorithm and the LC algorithm, the algorithm in this paper has a greater advantage in load balance degree. Similarly,

Table 1: Experimental configuration.

	Management node	Other nodes
CPU	2.3 GHz	2.3 GHz
RAM	32 GB	16 GB
Hard disk	1 TB	750 GB
Bandwidth	1 Gbps	1 Gbps

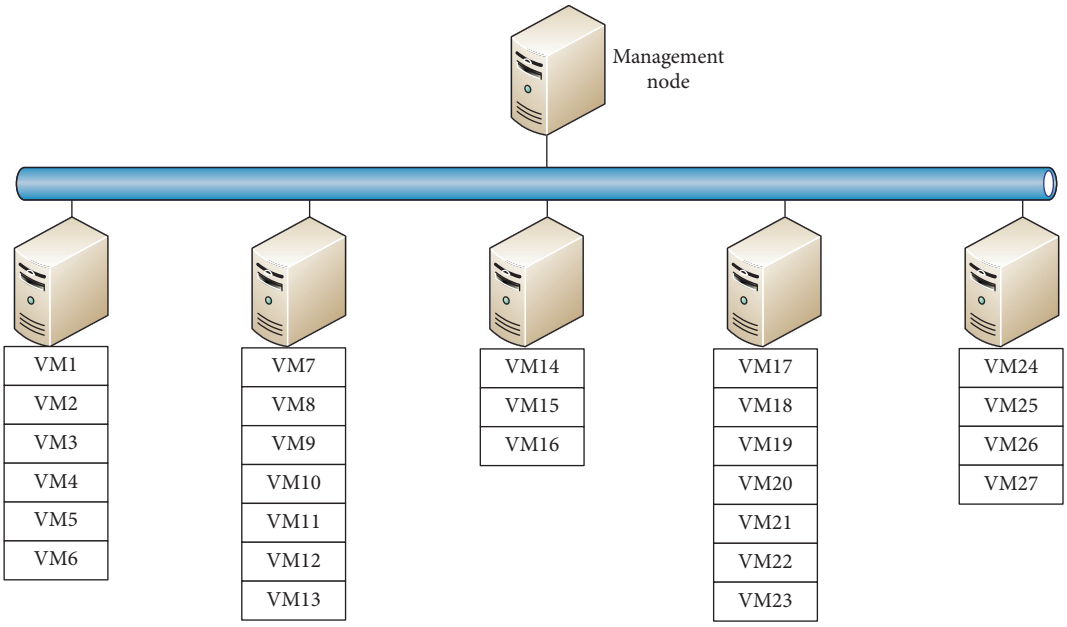

Figure 8: Experimental environment topology.

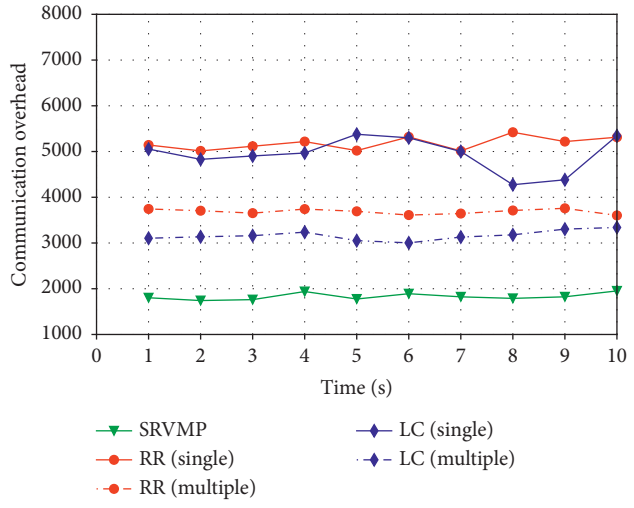

Figure 9: Comparison of communication overhead.

Figures 10(b) and 10(c) also prove that the SRVMP algorithm can achieve system load balancing. Since the AG algorithm has a higher balance degree, this paper will do an experiment to check the execution time of the task. The experimental result is shown in Figure 11.

It can be seen from Figure 11 that the SRVMP algorithm proposed in this paper has a greater advantage in task execution time. When the number of tasks is small, there is not much difference in the execution time, but when the number of tasks gradually increases, the execution time of the RR algorithm and the LC algorithm increases significantly. The time complexity of the SRVMP algorithm is $O(n^2)$, the time complexity of the ant colony algorithm is $O(nc \cdot n^3)$, and nc is the number of iterations of the ant

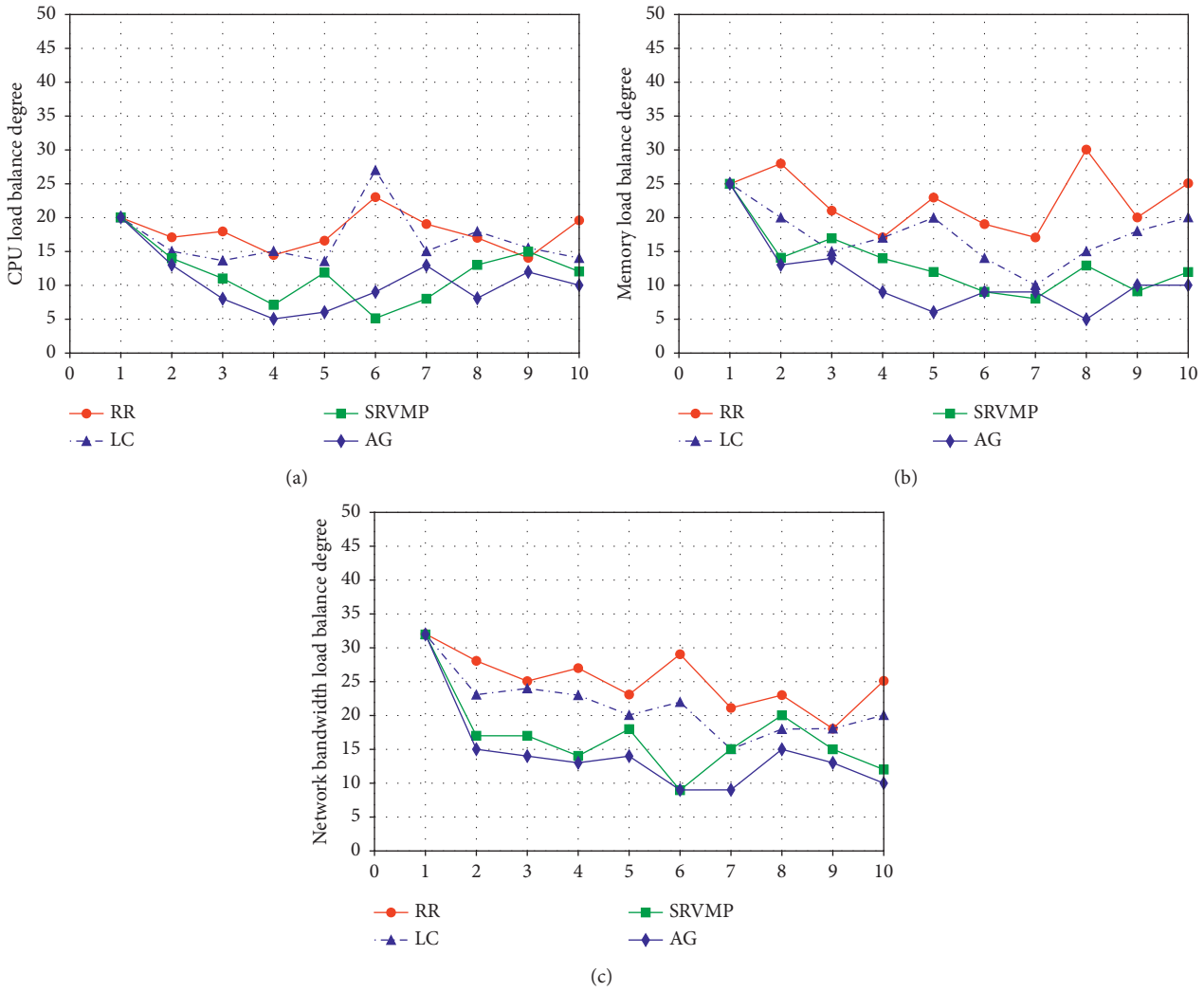

Figure 10: Comparison of load balance degree.

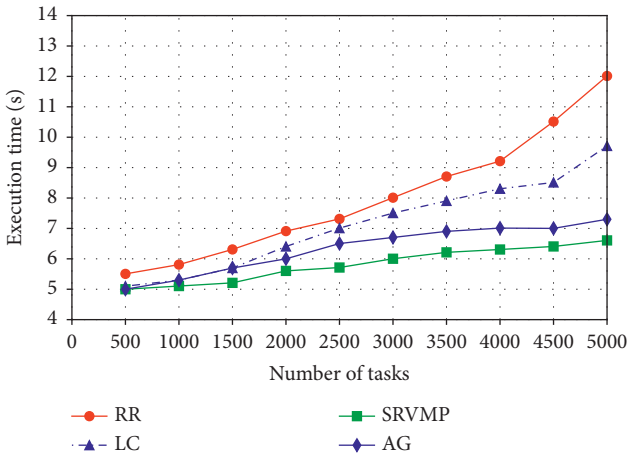

Figure 11: Comparison of task execution time.

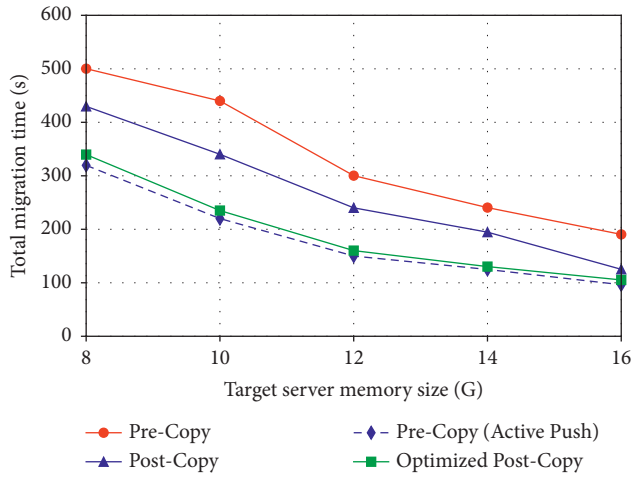

Figure 12: Comparison of total migration time (TMT).

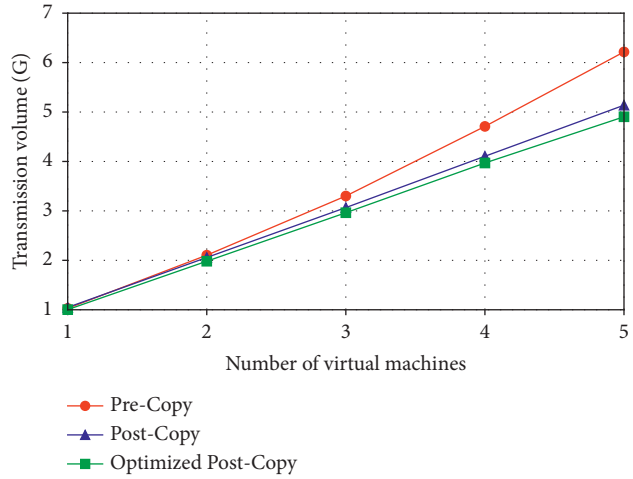

Figure 13: Comparison of total transferred data.

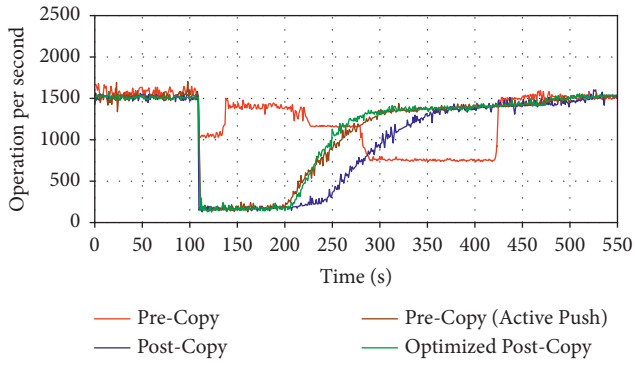

Figure 14: Comparison of operations per second.

colony algorithm. The AG algorithm requires multiple iterations. Therefore, although the SRVMP algorithm is slightly worse than the AG algorithm in system load balancing, it does not affect the task execution speed.

Experiment 3. The VM migration strategy based on service relevance designed in this paper uses an improved post-copy method (optimized post-copy) PCBSP to perform memory migration.

This experiment compares PCBSP with pre-copy and post-copy and uses total migration time (TMT), total migrated data, and page fault conditions as the criteria for verifying the efficiency of the method [34].

5.2.1. Total Migration Time (TMT). We set the target server's memory in the form of virtual memory in the form of 8G, 10G, 12G, 14G, and 16G and checked the changes of TMT in the four methods. The experimental results are shown in Figure 12.

Figure 12 shows that as the target server memory gets larger, the memory pressure gets smaller and smaller, and the TMT gap of each method gradually decreases. As can be seen from the figure, pre-copy takes the longest time, post-copy takes slightly shorter time, and post-copy (Active Push) further shortens the total time, because post-copy (Active Push) combines active push of memory and on-demand requests for memory to speed up the migration speed. The optimized post-copy method used in this article takes slightly longer than post-copy (Active Push). This is because the optimized post-copy method needs to first migrate the memory pages to TD and then to the target server.

5.2.2. Total Migrated Data. In this experiment, the original server and the target server were kept unchanged, and we compared the changes in the amount of migrated data, respectively, when the number of VMs is 1, 2, 3, 4, and 5. Since post-copy and post-copy (Active Push) have roughly the same amount of migrated data, the experiment compared the amount of migrated data by pre-copy, post-copy, and optimized post-copy methods. Figure 13 shows the comparison of the transferred data volume of the three methods. In the figure, the pre-copy transferred data volume is the largest, because the pre-copy method needs to repeatedly transfer the dirty pages synchronously, while the post-copy method only needs to migrate the memory page once. The optimized post-copy method takes into account the problem of data redundancy, so the amount of transferred data is the least, reducing the migration overhead, and effectively improving the efficiency of VM migration.

5.2.3. Page Fault Interruption. In order to show the trade-off between MT and migration performance, select a suitable VM to be migrated as the database server, query the database content through Yahoo Cloud Servicing Benchmark (YCSB) [35], and calculate each data based on the data calculated by YCSB. The number of operations per second is used to measure migration performance. This paper shows the changes in the number of operations per second of several methods over time. According to continuous monitoring of migration performance, the effect is shown in Figure 14.

The figure shows that, from about 110 seconds, the post-copy-related methods have always remained at a low level from the beginning of the migration. This is caused by a large number of page faults. At this time, the request for page faults and the target server receives memory pages to compete for traffic. The performance of the pre-copy method will also decline, but the decline is relatively gentle, because the pre-copy method does not have the effect of page faults. We compared it with the other two post-copy methods; the method in this paper recovers faster. Although the performance of the post-copy method decreases less, the MT is the longest. We consider that the application background of this paper is to complete the VMM as quickly as possible, so the trade-off between MT and migration performance should choose MT as a more important performance indicator when the migration performance is acceptable.

In summary, the post-copy memory migration method based on service priority proposed in this chapter can indeed reduce the VM migration time and reduce the occurrence of page faults when the migration performance is less affected; that is, it can quickly and efficiently complete VM migration.

6. Conclusion

Edge computing-based IoT provides a platform for data calculation and storage for the IoT devices. The data generated by IoT devices contains a large amount of user identity information, location information, and sensitive information. Therefore, data security and privacy issues in edge computing-based IoT are becoming increasingly prominent. Side-channel attacks steal the private information of other virtual machines by coresident virtual machines to bring huge security threats to edge computing. Virtual machine migration is the main way to defend against side-channel attacks. When selecting VM for migration, the degree of association between services is not considered. If some closely connected VMs are migrated to different servers, it will bring a lot of communication overhead during the calculation and migration process. In the process of VM memory migration, the pre-copy method is often used, which needs to repeatedly transfer a large number of dirty pages, thereby increasing the migration time of the virtual machine.

For the above problems, this paper proposes a VM migration strategy based on service relevance. First, define the service relevance factor to quantify the degree of relevance between services. Then, design a VM selection strategy based on service relevance, and group closely related VMs into a VM migration group to migrate. Finally, we design a post-copy memory migration method based on service priority (PCBSP); it can quickly migrate out of the VM migration group and effectively reduce the page fault rate. Through comparative experiments, it is verified that the VM selection method based on service relevance can effectively reduce the communication overhead in VM live migration. However, there are still two limitations in this paper. The next step of this paper is as follows: (1) in the VM migration, the downtime caused by server failure or overload is not considered. Once failure or downtime occurs, the VM migration will be invalid, and it is difficult to recover the VM after failure. The next step should consider how to deal with failure or downtime when using this method to migrate VMs. (2) PCBSP only calculates the service priority uniformly according to the dependency and dependent

relationship of the service, without considering the specific situation. In the next step, we should consider the specific situation to make the calculation of service priority more accurate and further reduce the page fault rate.

Data Availability

The experimental data of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest

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

Acknowledgments

This work was supported in part by the Major Projects of Natural Science Research in Universities in Jiangsu Province under Grant 17KJA413001.

References

[1] X.-J. Chen, B.-D. Chen, X.-M. Jiang, X.-B. Chen, and W.-H. Cai, "Improved cloud computing architecture for the Internet of Things," *Journal of Internet Technology*, vol. 17, no. 4, pp. 683–693, 2016.

[2] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE access*, vol. 6, pp. 6900–6919, 2017.

[3] B. Omoniwa, R. Hussain, M. A. Javed, S. H. Bouk, and S. A. Malik, "Fog/edge computing-based IoT (FECIoT): architecture, applications, and research issues," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4118–4149, 2018.

[4] M. S. Sheikh, J. Liang, and W. Wang, "Security and privacy in vehicular ad hoc network and vehicle cloud computing: a survey," *Wireless Communications and Mobile Computing*, vol. 2020, Article ID 5129620, 25 pages, 2020.

[5] C. Su and Q. Zeng, "Survey of CPU cache-based side-channel attacks: systematic analysis, security models, and countermeasures," *Security and Communication Networks*, vol. 2021, Article ID 5559552, 15 pages, 2021.

[6] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer And Communications Security*, L, USA, November 2009.

[7] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in *Proceedings of the International Conference on Cryptographic Hardware And Embedded Systems*, Santa Barbara, CA, August 2016.

[8] Y. Zhang, Y. Mao, M. Xu, F. Xu, and S. Zhong, "Towards Thwarting Template Side-Channel Attacks in Secure Cloud Deduplications," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 3, pp. 1008–1018, 2019.

[9] D. Park, G. Kim, D. Heo, S. Kim, H. Kim, and S. Hong, "Single trace side-channel attack on key reconciliation in quantum key distribution system and its efficient countermeasures," *ICT Express*, vol. 7, no. 1, pp. 36–40, 2021.

[10] C. Yang, Y.-f. Guo, H.-c. Hu, Y.-w. Wang, Q. Tong, and L.-s. Li, "Driftor: mitigating cloud-based side-channel attacks by switching and migrating multi-executor virtual machines," *Frontiers of Information Technology & Electronic Engineering*, vol. 20, no. 5, pp. 731–748, 2019.

[11] X. Wang, L. Wang, F. Miao, and J. Yang, "SVMDF:a secure virtual machine deployment framework to mitigate co-resident threat in cloud," in *Proceedings of the 2019 IEEE Symposium on Computers and Communications (ISCC)*, Barcelona, Spain, June 2019.

[12] S. B. Melhem, A. Agarwal, N. Goel, and M. Zaman, "A Markov-based prediction model for host load detection in live VM migration," in *Proceedings of the 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, Prague, Czech Republic, August 2017.

[13] S. Sotiriadis, N. Bessis, and R. Buyya, "Self managed virtual machine scheduling in cloud systems," *Information Sciences*, vol. 433-434, pp. 381–400, 2018.

[14] L. Liu, S. Zheng, H. Yu, V. Anand, and D. Xu, "Correlation-based virtual machine migration in dynamic cloud environments," *Photonic Network Communications*, vol. 31, no. 2, pp. 206–216, 2016.

[15] M. Rajabzadeh and A. T. Haghighat, "Energy-aware framework with Markov chain-based parallel simulated annealing algorithm for dynamic management of virtual machines in cloud data centers," *The Journal of Supercomputing*, vol. 73, no. 5, pp. 2001–2017, 2017.

[16] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, Hangzhou, China, December 2010.

[17] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, Heidelberg, Germany, December 2008.

[18] M. K. Gupta and T. Amgoth, "Resource-aware virtual machine placement algorithm for IaaS cloud," *The Journal of Supercomputing*, vol. 74, no. 1, pp. 122–140, 2018.

[19] R. Kanniga Devi, G. Murugaboopathi, and M. Muthukannan, "Load monitoring and system-traffic-aware live VM migration-based load balancing in cloud data center using graph theoretic solutions," *Cluster Computing*, vol. 21, no. 3, pp. 1623–1638, 2018.

[20] L. Lin, J. Chen, P. K. Kudjo, and O. Michael, "A novel routing protocol for content-based publish/subscribe model in mobile sensor networks," *International Journal of Distributed Sensor Networks*, vol. 15, no. 9, 2019.

[21] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A survey on virtual machine migration: challenges, techniques, and open issues," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1206–1243, 2018.

[22] U. Mandal, P. Chowdhury, M. Tornatore, C. U. Martel, and B. Mukherjee, "Bandwidth provisioning for virtual machine migration in cloud: strategy and application," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 967–976, 2016.

[23] M. R. Desai and H. B. Patel, "Efficient virtual machine migration in cloud computing," in *Proceedings of the 2015 15th International Conference On Communication Systems And Network Technologies*, Gwalior, India, April 2015.

[24] J. Zhang, F. Ren, R. Shu, T. Huang, and Y. Liu, "Guaranteeing delay of live virtual machine migration by determining and provisioning appropriate bandwidth," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2910–2917, 2015.

[25] S. Nathan, U. Bellur, and P. Kulkarni, "On selecting the right optimizations for virtual machine migration," *ACM SIGPLAN Notices*, vol. 51, no. 7, pp. 37–49, 2016.

[26] K. Su, W. Chen, G. Li, and Z. Wang, "Rpff: a remote page-fault filter for post-copy live migration," in *Proceedings of the 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, Chengdu, China, December 2015.

[27] G. Sun, D. Liao, V. Anand, D. Zhao, and H. Yu, "A new technique for efficient live migration of multiple virtual machines," *Future Generation Computer Systems*, vol. 55, pp. 74–86, 2016.

[28] Z. Lei, E. Sun, S. Chen, J. Wu, and W. Shen, "A novel hybrid-copy algorithm for live migration of virtual machine," *Future Internet*, vol. 9, no. 3, p. 37, 2017.

[29] U. Deshpande, D. Chan, S. Chan, K. Gopalan, and N. Bila, "Scatter-gather live migration of virtual machines," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 196–208, 2015.

[30] S. Yadav, "Comparative study on open source software for cloud computing platform: Eucalyptus, openstack and opennebula," *International Journal of Engineering Science*, vol. 3, no. 10, pp. 51–54, 2013.

[31] E. L. Hahne, "Round-robin scheduling for max-min fairness in data networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 7, pp. 1024–1039, 1991.

[32] X. Ren, R. Lin, and H. Zou, "A dynamic load balancing strategy for cloud computing Platform based on exponential smoothing forecast," in *Proceedings of the 2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, Beijing, China, September 2011.

[33] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *Proceedings of the 2013 8th International Conference on Computer Engineering & Systems (ICCES)*, Cairo, Egypt, November 2013.

[34] D. Malhotra, "A critical survey of virtual ,machine migration techniques in cloud computing," in *Proceedings of the 2018 First International Conference On Secure Cyber Computing And Communication (ICSCCC)*, Jalandhar, India, December 2018.

[35] V. Abramova, J. Bernardino, and P. Furtado, "Evaluating cassandra scalability with YCSB," in *Proceedings of the International Conference On Database And Expert Systems Applications*, Munich, Germany, September 2014.