This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing

RIAL: Resource Intensity Aware Load Balancing in Clouds

Haiying Shen*, *Senior Member, IEEE*

Abstract—To provide robust infrastructure as a service (IaaS), clouds currently perform load balancing by migrating virtual machines (VMs) from heavily loaded physical machines (PMs) to lightly loaded PMs. The unique features of clouds pose formidable challenges to achieving effective and efficient load balancing. First, VMs in clouds use different resources (e.g., CPU, bandwidth, memory) to serve a variety of services (e.g., high performance computing, web services, file services), resulting in different overutilized resources in different PMs. Also, the overutilized resources in a PM may vary over time due to the time-varying heterogeneous service requests. Second, there is intensive network communication between VMs. However, previous load balancing methods statically assign equal or predefined weights to different resources, which lead to degraded performance in terms of speed and cost to achieve load balance. Also, they do not strive to minimize the VM communications between PMs. We propose a Resource Intensity Aware Load balancing method (RIAL). For each PM, RIAL dynamically assigns different weights to different resources according to their usage intensity in the PM, which significantly reduces the time and cost to achieve load balance and avoids future load imbalance. It also tries to keep frequently communicating VMs in the same PM to reduce bandwidth cost, and migrates VMs to PMs with minimum VM performance degradation. We also propose an extended version of RIAL with three additional algorithms. First, it optimally determines the weights for considering communication cost and performance degradation due to VM migrations. Second, it has a more strict migration triggering algorithm to avoid unnecessary migrations while still satisfying Service Level Objects (SLOs). Third, it conducts destination PM selection in a decentralized manner to improve scalability. Our extensive trace-driven simulation results and real-world experimental results show the superior performance of RIAL compared to other load balancing methods.

1 Introduction

Cloud computing is becoming increasingly popular due to its ability to provide unlimited computing services with the pay-as-you-go model. Currently cloud systems employ virtualization technology to provide resources in physical machines (PMs) in the form of virtual machines (VMs). Users create VMs deployed on the cloud on demand. Each VM runs its own operating system and consumes resources (e.g., CPU, memory and bandwidth) from its host PM.

Cloud providers supply services by signing Service Level Agreement (SLA) with cloud customers that serves as both the blueprint and the warranty for cloud computing. Under-provisioning of resources leads to SLA violations while over-provisioning of resources leads to resource underutilization, and a consequent decrease in revenue for the cloud providers. Under this dilemma, it is important for cloud providers to fully utilize cloud resources and meanwhile uphold the SLAs. In order to provide robust infrastructure as a service (IaaS), clouds currently perform load balancing by migrating VMs from heavily loaded PMs to lightly loaded PMs so that the utilizations of PMs' resources (defined as the ratio between actual requested resource amount and the resource capacity) are below a threshold. Previously proposed load balancing methods [1]–[5] combine the utilizations of different resources in selecting VMs to migrate and finding the most suitable destination PMs. They predefine a weight (or give equal weight) for each resource, calculate the weighted product of different resource utilizations to represent the load of PMs and the weighted product of owned amount of each resource to represent the capacity of PMs, and then migrate VMs from the most heavily loaded PMs to the most lightly loaded PMs.

- * *Corresponding Author. Email: hs6ms@virginia.edu; Phone: (434) 924-8271; Fax: (434) 982-2214.*
- *Department of Computer Science, University of Virginia, Charlottesville, VA 22904-4740.*
  E-mail: hs6ms@virginia.edu

By assigning different resources equal or predefined weights, these methods neglect the unique feature of clouds of time-varying and different overutilized resources in different PMs. Cloud VMs use different resources to serve a variety of services (e.g., high performance computing, web hosting, file service), resulting in different overutilized resources and different resource intensities (e.g., CPU-intensive, MEM-intensive) in different PMs. *Resource intensity* here means the degree that a type of resource is demanded for services. By leveraging different resource intensities (e.g., moving a CPU-intensive and non-MEM-intensive VM from a CPU-intensive PM to a CPU-underutilized PM), we can more quickly achieve and more constantly retain the load balanced state with fewer VM migrations (i.e., fast and constant convergence). As cloud tasks are different from customers to customers and vary with time, the overutilized resources in a PM may vary over time. Predetermined or equal resource weight cannot adapt to the heterogeneous resource intensities among PMs and time-varying resource intensity in one PM.

For example, consider 4 PMs in Figure 1, where overloaded PM4 hosts 3 VMs. Because CPU is overutilized while MEM is underutilized in PM4, considering resource intensity, VM1 is the best option to move out since it has high consumption on high-intensity CPU and low consumption on low-intensity MEM. PM1 is the best option for the destination PM because it has most available CPU capacity for the CPU-intensive VM1. Without considering resource intensity, the previous methods may choose VM2 and VM3 to migrate out, and choose PM2 and PM3 as the destination PMs. Section 4.4 presents why considering intensity is better in detail with experiment measurement.

We aim to not only reduce the number of migrations in achieving the load balanced state but also avoid load imbalance in the future (i.e., fast and constant convergence) while minimizing the adverse effect of migration on the quality of cloud services. In addition to reducing load balancing cost, reducing VM migrations also mitigates the negative effect on cloud services because each migration i) generates a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing

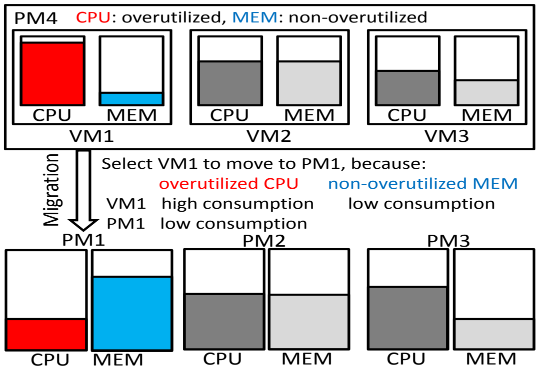


Fig. 1: Migration VM and destination PM selection.

service downtime; and ii) requires extra amount of network bandwidth and cache warm-up at the destination [6], [7].

In this paper, we propose a Resource Intensity Aware Load balancing method (RIAL). The advantages of RIAL are threefold. First, RIAL novelly distinguishes different PMs, different resource intensities and considers time-varying resource intensity in a PM when determining resource weights. For each PM, RIAL assigns different weights to different resources according to their intensities, which are then used in selecting VMs to migrate and finding destination PMs in each load balancing operation. Thus, an overloaded PM migrates out its VMs with high consumption on high-intensity resources and low consumption on low-intensity resources, hence quickly relieving its load while fully utilizing its resources. Also, the selected destination PM has high capacity on the high-intensity resources, which proactively avoids overloading destination PMs in the future. As RIAL determines the weight of a resource based on its current intensity, it is adaptive to dynamically changing resource intensities in different PMs. Second, RIAL selects migration VMs that have low communication rates with other VMs residing in the same PM in order to reduce bandwidth consumption. Communication rate between two VMs is the number of contacts between them in a unit time period. Third, when selecting destination PMs, RIAL tries to minimize the VM performance degradation due to migration. With the three advantages, RIAL achieves fast and constant convergence with fewer migrations while minimizing the interruption to cloud services.

We also propose an extended version of RIAL with three additional algorithms. First, it optimally determines the weights for considering communication cost and performance degradation due to VM migrations. Second, it has a more strict migration triggering algorithm to avoid unnecessary migrations while still satisfying Service Level Objects (SLOs). Third, it conducts destination PM selection in a decentralized manner to improve scalability.

We have conducted extensive trace-driven simulation and also deployed a small-scale cloud for real-world experiments. The experimental results show the superior performance of RIAL compared to other load balancing methods with fewer migrations, lower VM performance degradation and lower VM communication cost.

The rest of this paper is organized as follows. Section 2 briefly describes the related work. Section 3 presents the objective of RIAL. Section 4 first presents the detailed design of RIAL and an analysis of its performance compared to other load balancing methods, and then presents the three additional algorithms for the extended version of RIAL. Section 5 evaluates RIAL in both simulation and real-world experiments in comparison with other load balancing methods. Finally, Section 6 summarizes the paper with remarks on our future work.

2 Related Work

Many load balancing methods have been proposed to deal with PM overload problem using VM migration [1]–[5]. Sandpiper [1] tries to move load from the most overloaded servers to the most underloaded servers. It defines volume for VMs and PMs: $volume$=(1/(1-u_{cpu}))*(1/(1-u_{net}))*(1/(1-u_{mem})), where u is resource utilization. It also defines a volume-to-size ratio (VSR) for each VM: VSR=$volume/size$, where $size$ is the memory footprint of the VM. It then migrates the VM with the maximum VSR to the PM with the least volume. TOPSIS [5] predetermines weights for different criteria (e.g., CPU, memory, bandwidth, PM temperature). To select VMs to migrate (or select destination PM), it first forms a weighted normalized decision matrix with the utilizations of VMs of a PM (or PMs) with respect to each criterion. It then determines the ideal solution by using the maximum utilization for the benefit criteria and the minimum utilization for the cost criteria. Khanna *et al.* [4] treated different resources equally. They proposed to select the VM with the lowest product of resource utilizations from the overloaded PM and migrate it to the PM that has the least residual capacity big enough to hold this VM. Arzuaga *et al.* [2] used predetermined resource weights to calculate the product of weighted utilizations of different resources of a PM or a VM as its load. It then chooses the VM with the highest load from an overloaded PM to migrate to a selected PM that yields the greatest improvement of the system imbalance metric. Tang *et al.* [8] proposed a load balancing algorithm that strives to maximize the total satisfied application demand and balance the load across PMs. They define load-memory ratio of an instance as its CPU load divided by its memory consumption to measure its resource utilization. However, all previous methods statically assume equal or predefined weights for different resources, which may not be correct due to the different time-varying demands on different resources in each PM. RIAL is distinguished from these methods in that it dynamically determines the resource weight based on the demand on the resource in each PM, which leads to fast and constant convergence to the load balanced state.

Xu *et al.* [9] reviewed the state-of-the-art research on managing the performance overhead of VMs, and summarize them under diverse scenarios of the IaaS cloud, ranging from the single-server virtualization, a single mega datacenter, to multiple geodistributed datacenters. Li *et al.* [10] proposed effective VM placement methods to reduce the network cost in cloud data centers. Xu *et al.* [9], [11] proposed methods that consider VM performance (such as VM performance degradation caused by VM migration) when making the VM provisioning or migration decision. Lim *et al.* [12] modelled a migration process of a VM instance as a pair of jobs that run at the hosts of sender and receiver and proposed a method to analyze the migration time and the performance impact on multi-resource shared systems for completing given VM assignment plan. The novelty of RIAL compared to these works is the intensity-awareness,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing

which helps to reduce the number of VM migrations and maintain the load balanced state in the system.

Some works deal with load balancing on one resource such as storage [13] and bandwidth [14]–[16]. Hsiao *et al.* [13] proposed a load balancing algorithm for distributed file systems in clouds by moving file chunks from overloaded servers to lightly loaded servers. Oktopus [14] provides static reservations throughout the network to implement bandwidth guarantees. Popa *et al.* [16] navigated the trade-off space of requirements-payment proportionality, resource minimum guarantee and system utilization when sharing cloud network bandwidth. Xie *et al.* [15] proposed PROTEUS for bandwidth provisioning using predicted bandwidth utilization profile in order to increase the system bandwidth utilization and reduce the cost to the tenants. However, by focusing on only one resource, these approaches cannot be directly used for PM load balancing where VMs use different types of resources.

Many other works for resource management in clouds deal with scheduling incoming workload requests or initial placement of VMs with the concern of cost and energy efficiency [17]–[20]. Lin *et al.* [17] proposed an algorithm to achieve dynamic right-sizing in datacenters in order to save energy. It uses a prediction window of future arrivals to decide when to turn off an idle server. Maguluri *et al.* [18] focused on resource allocation that balances the load among servers to achieve throughput optimization. They considered a stochastic model of a cloud computing cluster, in which jobs arrive according to a stochastic process and request VMs. The authors showed that the widely-used Best-Fit scheduling algorithm is not throughput-optimal, and proposed alternatives to achieve optimal throughput. The goal of RIAL is not to achieve optimal throughput. Meng *et al.* [20] used traffic patterns among VMs to determine VM placement in order to improve network scalability. Shrivastava *et al.* [19] proposed AppAware that considers inter-VM dependencies and the underlying network topology to place VMs with intensive mutual communication in the same PM to reduce network traffic. Different from these two works, RIAL does not solely focus on improving network scalability or reduce network traffic, though it is one factor that RIAL considers in load balancing. Shen *et al.* [21] proposed an online resource demand prediction method to achieve adaptive resource allocation. Though resource demand prediction is out of the scope of this paper, this method can be used in RIAL to predict the future resource demands of VMs in load balancing to achieve the load balanced state in the future.

3 Objectives and Problem Statement

3.1 Notations and Final Objective

We consider a scenario in which a total of N PMs serve as a resource pool in the cloud. Let P_i denote PM i ($i = 1, 2, ..., N$), and n_i be the number of VMs hosted by P_i, denoted by V_{ij} ($j = 0, 1, ..., n_i$). Let C_{ik} ($k \in K$) denote the capacity (total amount) of type-k resource owned by P_i, where K is the set of resources.

Let $L_{ijk}(t)$ denote the type-k resource requested by V_{ij} in P_i at time t. It is a time varying function. To avoid small transient spikes of $L_{ijk}(t)$ measurements that trigger needless VM migrations, we use the average of $L_{ijk}(t)$ during time period Δt, denoted by $\overline{L_{ijk}}$.

$$\overline{L_{ijk}} = \frac{1}{\Delta t}\int_{t-\Delta t}^{t} L_{ijk}(t)\mathrm{d}t \tag{1}$$

Δt is an adaptive value depending on how fine grained we want to monitor the resource demands.

The usage of type-k resource in P_i is the sum of type-k resource requested by its VMs:

$$L_{ik} = \sum_{j=1}^{n_i} \overline{L_{ijk}} \tag{2}$$

Taking into account the heterogeneity of server capacities, we define the utilization rate of type-k resource in P_i (denoted by u_{ik}) as the ratio between actual requested resource amount of all VMs in P_i and the capacity of type-k resource of P_i.

$$u_{ik} = \frac{L_{ik}}{C_{ik}}. \tag{3}$$

We use Θ_k to denote the predetermined utilization threshold for the type-k resource in a PM in the cloud. The final objective of RIAL is to let each P_i maintain $u_{ik} < \Theta_k$ for each of its type-k resource (i.e., lightly loaded status). We call a PM with $u_{ik} > \Theta_k$ *overloaded PM*, and call this type-k resource *overutilized resource*.

Cloud customers buy VMs from cloud provider with predefined capabilities. For example, a small VM instance in Amazon EC2 is specified by 1.7GB of memory, 1 EC2 compute unit, 160GB of local instance storage, and a 32-bit platform. We use C_{ijk} to denote label capacity of V_{ij} corresponding to type-k resource. The utilization of V_{ij} is defined as

$$u_{ijk} = \frac{\overline{L_{ijk}}}{C_{ijk}} \tag{4}$$

In order to deal with heterogeneity, where the VM capacities are not the same, u_{ijk} can be defined in a new way: $\hat{u}_{ijk} = \frac{u_{ijk}\cdot C_{ijk}}{C_{ik}}$ or $\hat{u}_{ijk} = \frac{\overline{L_{ijk}}}{C_{ik}}$.

Like the load balancing methods in [1], [5], RIAL can use a centralized server(s) to collect node load information and conduct load balancing. It can also use a decentralized method as in [13] to conduct the load balancing. In this paper, we focus on how to select VMs and destination PMs to achieve a fast and constant convergence while minimize the adverse effect of VM migration on the cloud services.

3.2 Reducing VM Communications between PMs

The VMs belonging to the same customer are likely to communicate with each other much more frequently than with other VMs. Placing VMs with high communication frequency in different PMs will consume considerable network bandwidth. To save bandwidth consumption and hence increase cloud service quality, we try to keep VMs with frequent communication in the same PM. Thus, we try not to select VMs with a high communication rate with local VMs (residing in the same PM) to migrate to other PMs. We use T_{ijpq} to denote the communication rate between V_{ij} and V_{pq}, and use T_{ij} to denote the communication rate of V_{ij} with local VMs:

$$T_{ij} = \sum_{q=1}^{n_i} T_{ijiq} \tag{5}$$

Also, we try to choose the destination PM with the highest communication rate with migration VM V_{ij}. We denote the communication rate between V_{ij} and PM P_p as

$$T_{ijp} = \sum_{q=1}^{n_p} T_{ijpq} \tag{6}$$

where n_p is the number of VMs in P_p.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing

3.3 Reducing VM Performance Degradation by Migrations

When a VM is being migrated to another PM, its performance (response time) is degraded [22]. We also aim to minimize the VM performance degradation caused by migrations. We calculate the performance degradation of VM V_{ij} migrating to PM P_p based on a method introduced in [22], [23]:

$$D_{ijp} = d_{ip} \cdot \int_{t}^{t+\frac{M_{ij}}{B_{ip}}} u_{ij}(t)\mathrm{d}t \tag{7}$$

where t is the time when migration starts, M_{ij} is the amount of memory used by V_{ij}, B_{ip} is the available network bandwidth, $\frac{M_{ij}}{B_{ip}}$ indicates the time to complete the migration, $u_{ij}(t)$ is the CPU utilization of V_{ij}, and d_{ip} is the migration distance from P_i to P_p. The distance between PMs can be determined by the cloud architecture and the number of switches across the communication path [16], [20].

3.4 Problem Statement

In a cloud system, we denote the set of all overload PMs by $\mathcal{O}$ and the set of all lightly loaded PMs by $\mathcal{L}$. Given $\mathcal{O}$ and $\mathcal{L}$, our objective is to select V_{ij} from $P_i \in \mathcal{O}$ and then select the destination $P_p \in \mathcal{L}$ to migrate V_{ij} to in order to eliminate overloaded PMs and meanwhile minimize the number of VM migrations, the total communications between the migration VMs and PMs and the total performance degradation of all migration VMs. We use $\mathcal{S}_i$ to denote the set of selected migration VMs in P_i, and use $|\cdot|$ to represent the size of a set. Then, our problem can be expressed as:

$$\min |\{V_{ij} | V_{ij} \in \mathcal{S}_i,\ P_i \in \mathcal{O}\}| \tag{8}$$

$$\min \sum T_{ijp} \tag{9}$$

$$\min \sum D_{ijp} \tag{10}$$

$$subject\ to:\ u_{ik} \leq \Theta_k,\ \forall\, i, k \tag{11}$$

Our problem of VM migration is a variant of the multiple knapsack problem, which is NP-complete [24]. A simpler formulation of our problem has been shown to be NP-complete in [19], [20]. Our problem differs from them mainly in that it minimizes the number of VM migrations. We can construct a special instance of our problem that is similar to them and hence prove that our VM migration problem is NP-complete. We will present a method for solving this problem below.

4 The Design of RIAL

Like all previous load balancing methods, RIAL periodically finds overloaded PMs, identifies the VMs in overloaded PMs to migrate out and identifies the destination PMs to migrate the VMs to. In RIAL, each PM P_i periodically checks its utilization for each of its type-k ($k \in K$) resources to see if it is overloaded. We use L and O ($L \cup O = K$) to denote the set of resource types in the PM that are non-overutilized and overutilized, respectively. An overloaded PM triggers VM migration to migrate its VMs to other PMs until its $u_{ik} \leq \Theta_k$ ($k \in K$). Below, we present the methods for selecting VMs to migrate and for selecting destination PMs with the objectives listed in Section 3.4.

4.1 Selecting VMs to Migrate

We first introduce a method to determine the weight of each type of resource based on resource intensity. We aim to find VMs to migrate out of each overloaded P_i to quickly reduce its workload. If P_i is overutilized in CPU, then we hope to select the VM with the highest CPU utilization in order to quickly relieve P_i's load. Since non-overutilized resources do not overload P_i, we do not need to reduce the utilization of these resources in P_i. Therefore, we also aim to select the VM with the lowest utilization in non-overutilized resources in order to fully utilize resources. To jointly consider these two factors, we determine the weight for each type-k resource according to its overload status in P_i.

To achieve the above mentioned objective, we give overutilized resources relatively higher weights than non-overutilized resources. Among the non-overutilized resources, we assign lower weights to the resources that have higher utilizations in order to more fully utilize resources in the PM. Therefore, the weight for a non-overutilized resource with resource utilization u_{ik} is determined by

$$w_{ik} = 1 - u_{ik}.$$

A resource with utilization zero receives a weight of 1. The weight decreases as the utilization increases. The resource with a utilization closest to the threshold Θ_k (i.e., $u_{ik} < \Theta_k$ and $u_{ik} \approx \Theta_k$) receives the lowest weight $1 - \Theta_k$. Thus, this resource has the lowest probability to be migrate out.

Among the overutilized resources, the resources that have higher utilizations should receive higher weights than those with relatively lower utilizations. For the overutilized resources that have similar but different utilization values, we hope to assign much higher weights to the resources with higher utilizations and assign much lower weights to the resources with lower utilization. That is, we exaggerate the difference between the weights of resources based on the difference between their utilization. Thus, we use a power function with a basic form to determine the weight for an overutilized resource with resource utilization u_{ik}:

$$w_{ik} = \frac{1}{a u_{ik}^{\alpha} + b},$$

where a and b are constant coefficients, and α is an integer exponent. In order to simplify the above equation and at the same time meet the design requirements as discussed previously, we let $\alpha = 1$. To satisfy the monotonically increasing property (i.e., higher utilization receives higher weight), we set $a = -1$. Considering that the domain of the function should cover [Θ_k,1) (i.e., for an overutilized resource, $\Theta_k \leq u_{ik} < 1$), so $b = 1$. As a result, the weight given to a resource can be determined by

$$w_{ik} = \begin{cases} \frac{1}{1-u_{ik}}, & \text{if } k \in O, \\ 1 - u_{ik}, & \text{if } k \in L. \end{cases} \tag{12}$$

The weight of resource k (w_{ik}) means the priority of migrating this resource out. The function in Equation 12 is shown in Figure 2. That is, for an overutilized resource $k \in O$ ($u_{ik} \geq \Theta_k$), a higher utilization leads to a higher weight. For a non-overutilized resource $k \in L$ ($u_{ik} < \Theta_k$), a higher utilization leads to a lower weight. Note that $w_{ik} > 1$ for a resource $k \in O$ al-

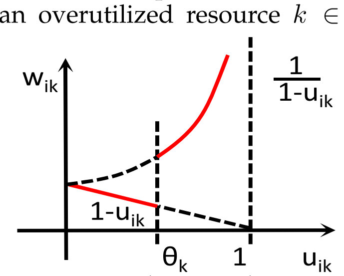


Fig. 2: Weight vs. utilization.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing

ways has a higher weight than $w_{ik} < 1$ for a resource $k \in L$, which means that overutilized resources always have higher priority to migrate out than underutilized resources. The figure shows that, determining resource weight w_{ik} based on Equ. (12) satisfies all the requirements discussed before. For example, when $u_{ik}<\Theta_k$, $w_{ik}=1-u_{ik}$ is a decreasing function with a constant slope (left red curve) of -1. When $u_{ik} \geq \Theta_k$, $w_{ik}=\frac{1}{1-u_{ik}}$ is an increasing function with increasing slopes (right red curve). $w_{ik}>1$ for an overutilized resource ($u_{ik}\geq\Theta_k$) while $w_{ik}<1$ for a non-overutilized resource ($u_{ik}<\Theta_k$). The resource with a utilization smaller than and close to the threshold has the lowest weight.

We use the Mullti-Criteria Decision Making (MCDM) [25] method to select the VM to migrate. Basically, the MCDM method calculates the weighted distances of all the candidates from the ideal solution, and selects the one with shortest distance. Recall that u_{ijk} is the type-k resource utilization rate of VM V_{ij}. Using the MCDM method, we establish a $|K| \times n_i$ decision matrix D_i for PM P_i with n_i VMs as

$$D_i = \begin{pmatrix} u_{i11} & \cdots & u_{in_i1} \\ \vdots & \ddots & \vdots \\ u_{i1|K|} & \cdots & u_{in_i|K|} \end{pmatrix} \tag{13}$$

in which each row represents one type of resource and each column represents each VM in P_i. In the case of heterogenous VM types, we use the normalized VM utilizations and simply replace u_{ijk} with $\hat{u}_{ijk}$ in Equ. (13).

We then normalize the decision matrix:

$$X_i = \begin{pmatrix} x_{i11} & \cdots & x_{in_i1} \\ \vdots & \ddots & \vdots \\ x_{i1|K|} & \cdots & x_{in_i|K|} \end{pmatrix} \tag{14}$$

where

$$x_{ijk} = \frac{u_{ijk}}{\sqrt{\sum_{j=1}^{n_i} u_{ijk}^2}} \tag{15}$$

Next, we determine the ideal migration VM (denoted by R_{VM}) which has the highest usage of overutilized resources and has the lowest usage of non-overloaded resources. That is,

$$R_{VM} = \{r_{i1}, ..., r_{i|K|}\} = \{(\max_j x_{ijk}|k \in O), (\min_j x_{ijk}|k \in L)\}; \tag{16}$$

for each type-k resource, if it is overutilized, its r_{ik} is the largest element from $(x_{i1k} \cdots x_{ijk} \cdots x_{in_ik})$ in X_i; otherwise, r_k is the smallest element.

As indicated in Section 3.2, we also hope to select the VM with the lowest communication rate to other VMs in the same PM (i.e., T_{ij}) in order to reduce subsequent VM communication cost after migration. Therefore, we set the ideal value of T_{ij} to 0. We then calculate the Euclidean distance of each candidate V_{ij} in P_i with the ideal VM and ideal T_{ij}.

$$l_{ij} = \sqrt{\sum_{k=1}^{|K|}[w_{ik}(x_{ijk} - r_{ik})]^2 + [w_t T_{ij}]^2}, \tag{17}$$

where w_t is the weight of the communication rate and it can be adaptively adjusted based on the tradeoff between the convergence speed/cost and the network bandwidth cost for VM communication. The migration VM is the VM with the shortest Euclidean distance (l_{ij}), i.e., the most similar resource utilizations as the ideal VM. After selecting a VM V_{ij}, RIAL checks if V_{ij}'s $u_{ijk}(k \in K)$ is in R_{VM}. If so, RIAL replaces V_{ij}'s u_{ijk} in R_{VM} with the updated value. RIAL then continues to choose the VM with the second shortest l_{ij}. Using the above method, RIAL keeps selecting migration VMs from P_i until P_i is no longer overloaded.

4.2 Selecting Destination PMs

When selecting destination PMs to migrate the selected VMs from P_i, we consider resource intensity, VM communication rate and performance degradation as indicated in Section 3. We use J to denote the set of lightly loaded PMs. We also use the MCDM method for destination PM selection. We build the $|K| \times |J|$ decision matrix D' as

$$D' = \begin{pmatrix} u_{11} & \cdots & u_{|J|1} \\ \vdots & \ddots & \vdots \\ u_{1|K|} & \cdots & u_{|J||K|} \end{pmatrix} \tag{18}$$

in which each row represents one type of resource and each column represents each lightly loaded PM.

We then normalize the decision matrix:

$$X' = \begin{pmatrix} x'_{11} & \cdots & x'_{|J|1} \\ \vdots & \ddots & \vdots \\ x'_{1|K|} & \cdots & x'_{|J||K|} \end{pmatrix} \tag{19}$$

where

$$x'_{jk} = \frac{u_{jk}}{\sqrt{\sum_{j=1}^{|J|} u_{jk}^2}} \tag{20}$$

Recall that the weight of type-k resource (w_{ik}) represents the priority of migrating this resource out from overloaded PM P_i. Hence, it also indicates the priority of considering available resource in selecting destination PMs. Therefore, we also use these weights for different resources in candidate PMs in order to find the most suitable destination PMs that will not be overloaded by hosting the migration VMs. We represent the ideal destination PM as

$$R'_{PM} = \{r'_1, ..., r'_k, ..., r'_{|K|}\} = \{\min_j x'_{jk}|k \in K\}. \tag{21}$$

consisting of the lowest utilization of each resource from the candidate PMs.

When choosing destination PMs, we also hope that the VMs in the selected destination PM P_p have higher communication rate with the migration VM V_{ij} (i.e., T_{ijp}) in order to reduce network bandwidth consumption. Thus, we set the ideal T_{ijp} to be the maximum communication rate between V_{ij} and all candidate PMs, $T_{max} = \max T_{ijp}$ ($p \in J$). Further, the performance degradation of the migrated VMs should be minimized.

By considering the above three factors, we calculate the Euclidean distance of each candidate PM P_p from the ideal PM.

$$l_{p,ij} = \sqrt{\sum_{k=1}^{|K|}[w_{ik}(x'_{pk} - r'_k)]^2 + [w_t(T_{ijp} - T_{max})]^2 + [w_d D_{ijp}]^2} \tag{22}$$

where w_d is the weight of performance degradation consideration that can be adaptively adjusted like w_t. We then select the PM with the lowest $l_{p,ij}$ value as the migration destination of selected VMs. If the selected PM does not have sufficient available resources to hold all VMs, the PM with the second lowest $l_{p,ij}$ is selected using the same method as selecting migration VMs. This process is repeated until the selected PMs can hold all selected migration VMs of P_i. Note that the magnitudes of w_t and w_d should be properly determined based on the practical requirements of the cloud on the tradeoff of the number of VM migrations, bandwidth cost and VM performance degradation. Higher w_t and w_d lead to more VM migrations, while lower w_t generates higher bandwidth cost for VM communications and lower w_d generates higher VM performance degradation. How to determine these magnitudes for an optimal tradeoff is left as our future work. Note that the selection of migration VMs is always before the allocation of destination PMs because this way, the best-fit destination PM can be selected for each selected migration VM. In this paper, we aim to avoid overloading PMs and meanwhile consider several factors (such as reducing PM communication rate) in achieving the goal (i.e., selecting migration VMs and destination PM). Sometimes, the VM migration may degrade the performance in terms of the considered factors such as increasing PM communication rate. Since avoiding overloaded PMs is the goal of our work, it has the highest priority compared with the considered factors. Our experimental results in Section 5 shows that the execution time of RIAL is acceptable. We will further improve RIAL to reduce the execution time in a very large scale system in our future work.

4.3 Parameter Determination

Our load balancing algorithm selects VMs to be migrated out from each overloaded PM and selects the destination PM to host each migrated VM in order to quickly reach the load balanced state in the system (i.e., quick convergence). Equ. (17) is used to select VMs that should be migrated out from an overloaded PM considering the weights for resources (w_{ik}) and for communication cost (w_t). Equ. (22) is used to select the destination PM considering the weights for resources (w_{ik}), for communication cost (w_t) and for performance degradation due to migration (w_d). The values of these weight parameters have a direct impact on the performance of our proposed load balancing algorithm. In this section, we present how to determine these parameters to achieve better performance.

As indicated in Equ. (17), in order to calculate the Euclidean distance of candidate VM V_{ij} when selecting a VM to migrate, we must determine w_{ik} and w_t. Recall that w_{ik} is determined by Formula 12. Then, we must first determine the value of w_t before we calculate the Euclidean distance The importance of considering the communication cost (w_t) should not overtake the importance of relieving overutilized resources (w_{ik}), which is the primary objective of our load balancing algorithm. A high w_t may lead to the failure of mitigating the load of overloaded resources, while a low w_t may lead to the unawareness of the communication rate in migration VM selection. Thus, in load balancing, we give the highest priority to offloading the excess load in an overloaded PM, and paying as much attention as possible to communication rates between VMs in order to maximize the VM communications within a PM.

Therefore, we determine w_t so that one of the VMs that are the most similar to the ideal VM without considering the communication cost is selected and at the same time w_t is maximized. Suppose V_{is} is the selected VM in the VM selection algorithm without considering the communication rate of the VMs (i.e., $w_t = 0$ in Equ. (17)):

$$V_{is} = \arg\min_{V_{ij}} l_{V_{ij}} \tag{23}$$

and

$$l_{V_{ij}} = \sqrt{\sum_{k=1}^{|K|} [w_{ik}(x_{ijk} - r_{ik})]^2} \tag{24}$$

A VM V_{ij} is regarded as one of the most similar VMs to the ideal VM, if

$$l_{V_{im}} \leq l_{V_{is}} + \delta_v, \tag{25}$$

where δ_v is a positive constant. By selecting a similar VM rather than the most similar VM without considering the communication cost (i.e., V_{is}), we slightly sacrifice the priority of offloading the excess load to reducing communication cost. The value of δ_v determines the extent of the sacrifice. We denote the set of VMs that satisfy Equ. (25) as $\mathcal{S}_v$. With our determined w_t, the VM in $\mathcal{S}_v$ that can maximally reduce the communication cost will be selected to migrate out. In the following, we explain how to determine the value of w_t based on δ_v and w_{ik} for the aforementioned objective.

The problem of finding the maximum w_t with the constrain of δ_v can be expressed as follows. Given the normalized decision matrix X_i of P_i and the ideal migration VM R_{VM}, the problem is to maximize w_t, subject to:

$$l_{im} \leq l_{ij}, \quad \forall\, V_{im} \in \mathcal{S}_v,\ V_{ij} \notin \mathcal{S}_v \tag{26}$$

where l_{im} and l_{ij} are calculated by Equ. (17). It means that V_{im} will always be selected to migrate out even with the maximized w_t. It is to ensure that the selected VM without considering the communication rate (Equ. (25)) will not change when taking into account the communication rate (Equ. (26)).

In order to solve this problem, we can combine Equ. (17) and Equ. (26), and then derive Equ. (27) below:

$$\sum_{k=1}^{|K|} w_{ik}^2[(x_{imk}-r_{ik})^2-(x_{ijk}-r_{ik})^2] \leq\ w_t^2(T_{ij}^2-T_{im}^2) \tag{27}$$

Since x_{ijk} is known, we can find x_{isk} and hence x_{imk} based on Equ. (23) and Equ. (25). Since T_{ij} and T_{im} are also known, we can solve Equ. (27). Equ. (27) can be solved based on the values of $T_{ij}^2 - T_{im}^2$ and $(x_{imk} - r_{ik})^2 - (x_{ijk} - r_{ik})^2$, which can be either positive or negative. We ignore useless constraints of these two values that are derived from the condition in Equ. (27). For example, if $T_{ij}^2 - T_{im}^2 >$ and $(x_{imk}-r_{ik})^2-(x_{ijk}-r_{ik})^2 < 0$, we derive that w_t is greater than a negative value, which is always true and thus useless. Then, we derived that when $(x_{imk}-r_{ik})^2-(x_{ijk}-r_{ik})^2 < 0$ and $T_{ij}^2 - T_{im}^2 < 0$,

$$w_t \leq \sqrt{\frac{\sum_{k=1}^{|K|} w_{ik}^2[(x_{imk} - r_{ik})^2 - (x_{ijk} - r_{ik})^2]}{T_{ij}^2 - T_{im}^2}} \tag{28}$$

Finally, we solve Equ. (28) and select the maximum value for w_t.

Solving Equ. (28) involves complicated calculations including determining weights for resources based on Equ. (12), finding V_{is} based on Equ. (23) and solving Equ. (27). In the following, we try to simplify the process of determining w_t. Since we consider mitigating the load of the overutilized resources and at the same time maximizing the VM communications within a PM (by selecting the VM that has minimal communications with the co-locating VMs to migrate out), we can further loose Equ. (28) to simplify the process of w_t determination. Specifically, we only consider the most sensitive weight, which is defined as the minimum weight of the overutilized resources:

$$w_m = \min\{w_k | k \in O\}. \tag{29}$$

Because w_m is the minimum weight of the overutilized resources, by ensuring that w_t does not overtake w_m, we can satisfy the condition that w_t does not overtake all the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing

weights of the overutilized resources with a high probability. We then determine w_t based on w_m in order to prevent w_t from overtaking the minimum weight of the overloaded resources. We aim to maximize w_t while guaranteeing that the most similar VM should be selected. To simplify the process, we can only consider the VM that is the most similar to the ideal VM and the VM that is the second similar. Because every VM together with the VM that is the most similar to the ideal VM can specify a range for the value of w_t, and the constraints placed on w_t by other VMs are relatively looser compared to the second similar VM. Suppose there is only one resource overutilized, and the weight is w_m; the VM (VM_0) which is the most similar to the ideal VM has normalized utilization x_0 and communication rate T_0; the VM (VM_1) which is the second similar has utilization x_1 and communication rate T_1. We use a linear function $l = w_m x + w_t T$ to represent Equ. (17). Then, the above problem can be expressed as: to maximize w_t, subject to

$$w_m x_0 + w_t T_0 \leq w_m x_1 + w_t T_1, \quad x_0, x_1, T_0, T_1 > 0. \tag{30}$$

Finally, we can find the maximum w_t as

$$w_t = -\frac{x_0 - x_1}{T_0 - T_1} w_m \tag{31}$$

In order to further make the determination of w_t easier, we derive a constant weight. As a rule of thumb, w_t is greater than 1, which is the maximal weight for a non-overutilized resource, because considering communication rate is more important than considering the non-overutilized resources. Also, weight w_t should be lower than the weight of overutilized resources, because mitigating the load on overload resources has the highest priority. That is, $w_t < \frac{1}{1-\Theta_k}$ based on Equ. (12). For example, for a threshold $\Theta_k = 0.75$, the weight for communication rate $w_t < 4$. Then, w_t can be set to constant 3, which is the maximum value that satisfies < 4. In our experiment in Section 5, with $\Theta_k = 0.75$, we set a constant to w_t, i.e., $w_t = 3$.

Next, we discuss how to determine the weight for communication cost (w_t) and for performance degradation due to migration (w_d) in Equ. (22) for the destination PM for a migrated VM. Different from VM selection, here, we need to determine two parameters. However, a formulated problem can only be used for optimizing one object. We then combine w_t and w_d to one optimization object. Then, similar to what has been discussed before, we can derive both w_t and w_d together for PM selection by altering the object function of the aforementioned problem for VM selection. That is, we place equal importance on the two weights since both weights are important (i.e., $w_d = w_t$) and try to maximize w_d. Suppose P_s is the selected destination PM in PM selection algorithm without considering the communication rate or performance degradation of the VMs (i.e., $w_t = 0$ and $w_d = 0$ in Equ. (22)):

$$P_s = \arg\min_{P_p} l_{P_p} \tag{32}$$

and

$$l_{P_p} = \sqrt{\sum_{k=1}^{|K|} [w_{ik}(x'_{pk} - r'_k)]^2} \tag{33}$$

A PM P_p is regarded as one of the most similar PMs to the ideal PM, if

$$l_{P_m} \leq l_{P_s} + \delta_p, \tag{34}$$

where δ_p is a positive constant. Similarly, δ_p represents the extent of the sacrifice of the priority of offloading overloaded resource to reducing communication cost and performance degradation due to VM migrations. We denote the set of PMs that satisfy Equ. (34) as $\mathcal{S}_p$. Then, the problem can be transformed to maximize w_t, subject to:

$$l_{m,ij} \leq l_{p,ij}, \quad \forall\, P_m \in \mathcal{S}_p,\ P_p \notin \mathcal{S}_p \tag{35}$$

Similarly, we can derive

$$w_t \leq \sqrt{\frac{\sum_{k=1}^{|K|} w_{ik}^2[(x'_{mk} - r'_k)^2 - (x'_{pk} - r'_k)^2]}{[(T_{ijp} - T_{max})^2 - (T_{imp} - T_{max})^2] + (D_{ijp}^2 - D_{imp}^2)}} \tag{36}$$

For more simplified w_t and w_d, we adopt $w_t = 3$ and $w_d = 3$ as the constant values for these weights. Similar as previous, for a threshold $\Theta_k = 0.75$, the weight for communication rate $w_t < 4$, the weight for performance degradation $w_d < 4$. Then, both w_t and w_d can be set to constant 3. We will show the experiment results with varying w_t and w_d in Section 5.

4.4 Performance Comparison Analysis

Compared to Sandpiper [1] and TOPSIS [5], RIAL produces fewer migrations. Because RIAL determines the resource weight based on resource intensity, it can quickly relieve overloaded PMs by migrating out fewer VMs with high usage of high-intensity resources. Also, the migration VMs have low usage of low-intensity resources, which helps fully utilize resources and avoids overloading other PMs. In addition, the migration destination has a lower probability of being overloaded subsequently as it has sufficient capacity to handle the high-intensity resources. Finally, RIAL leads to fewer VM migrations in a long term.

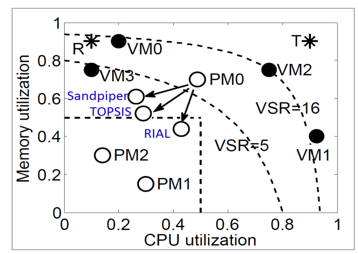
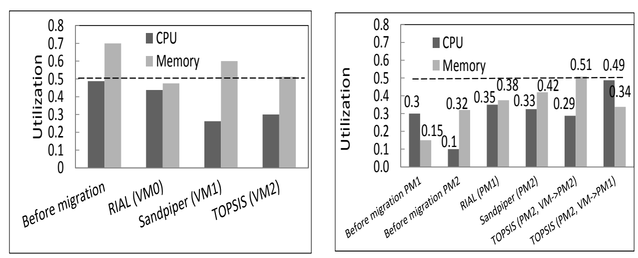


Fig. 3: VM and PM selection process.



(a) Utilizations of PM0 with different selected migration VMs.

(b) Utilizations of different selected destination PMs.

Fig. 4: Advantage of RIAL in reducing migrations.

We use an example with 3 PMs (PM0, PM1, PM2) to demonstrate the advantage of RIAL. In practice, the overloaded threshold should be close to 1. To make the example simple with few VMs, we set the threshold to 0.5, and only consider the CPU and memory resources. We assume that PM0 has 4 VMs (VM0, VM1, VM2, VM3) with the same capacity and PM0's capacity is four times of the VM's. PMs have the same capacity. As in [5], the weight of CPU and memory in TOPSIS is 9 and 4, respectively. Figure 3 shows the CPU and memory utilizations of the 4 VMs, VM0(0.2,0.9), VM1(0.9,0.4), VM2(0.75,0.75), VM3(0.1,0.75) and the 3 PMs, PM0(0.49,0.7), PM1(0.3,0.15), PM2(0.1,0.32). PM0 is overloaded in memory resource usage since 0.7>0.5.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing

TABLE 1: Number of migrations needed for load balance

	Sandpiper	TOPSIS	RIAL
# selected migration VMs	2	2	1
# of overload destination PMs after VM migrations	0	1	0
Total # of migrations	2	3	1

Sandpiper attempts to migrate the VM with maximum VSR=$volume/size$, where $volume=(1/(1-u_{cpu}))*(1/(1-u_{net}))*(1/(1-u_{mem}))$. Based on this formula, we draw two dash curves in Figure 3 to indicate the points whose VSR equals to 5 and 16, respectively. We see that among the 4 VMs, VM1 located beyond the curve of VSR=16 has the highest VSR. Therefore, Sandpiper selects VM1 to migrate out of PM1. TOPSIS first determines its ideal VM (T* in Figure 3) with the maximum CPU and memory utilizations from the 4 candidate VMs (i.e., (0.9, 0.9)), then compares the weighted distances of the 4 VMs to the ideal VM, and finally chooses VM2 that has the shortest distance. In RIAL, according to Equ. (16), the CPU and memory utilizations of the ideal VM (R* in Figure 3) are 0.1 and 0.9. Base on Equ. (12), the weights for memory and CPU are 3.33 and 0.51, respectively. Unlike TOPSIS, RIAL gives a weight to CPU smaller than memory, since CPU is not so intensively used as memory. RIAL finally chooses VM0 which has the shortest weighted distance to the ideal VM.

Figure 4(a) shows the CPU and memory utilizations of PM0 before VM migration and after migrating VM0, VM1 and VM2 by RIAL, Sandpiper and TOPSIS, respectively. The arrows in Figure 3 indicate the resource utilizations of PM0 after migration in each method, respectively. We see that neither migrating VM1 (by Sandpiper) nor migrating VM2 (by TOPSIS) can eliminate memory overload in PM0. Hence, these two methods require another migration. RIAL reduces both CPU and memory utilizations below the threshold.

For destination PM selection, PM1(0.3,0.15) and P-M2(0.1,0.32) are two candidates for the VM from PM0. Sandpiper selects the PM that has the least $volume$ as the destination, which is PM2. TOPSIS determines the ideal PM with the least CPU and memory utilization of all candidate PMs (i.e., (0.15, 0.1)), and selects the one with the shortest weighted distance to the ideal PM, which is PM2. However, after migrating VM2 to PM2, the memory utilization of PM2 increases to 0.51, higher than the threshold. Then, TOPSIS has to execute another migration and chooses PM1 to migrate VM2 to. RIAL determines the same ideal PM as TOPSIS, but assigns higher weight to memory, so it chooses PM1 as the destination that has the shortest weighted distance.

Figure 4(b) shows the CPU and memory utilizations of the destination PMs before and after migrations. TOPSIS overloads the destination PM2 in memory and needs another migration (VM2→PM1) to relieve its memory load. Though all three methods finally eliminate the memory overload in PM0, RIAL generates a more balanced state since resource utilizations after balancing are relatively lower than those in Sandpiper and TOPSIS, which reduces the probability of overloading PMs, and hence helps maintain the system load balanced state for a longer time period.

Table 1 lists the number of selected VMs to relieve overloaded PM0, the number of overloaded destination PMs after the VM migrations, and the total number of migrations to achieve the load balanced state in one load balancing operation. We see that RIAL generates the least number of migrations due to its advantages mentioned previously.

4.5 When to Trigger VM Migration

In today's cloud datacenter, VMs may generate transient workload spikes in PMs [26], which are sharp rises in the resource utilization that immediately followed by decreases. A PM may become overloaded (i.e., $u_{ik} > \Theta_k$) during a spike, and becomes underloaded after the spike. In this case, simply triggering the VM migration upon the observation that the resource utilization of a PM exceeds a threshold (i.e., $u_{ik} > \Theta_k$) generates unnecessary VM migration operations and overhead, and also fail to fully utilize resources. The occurrence of $u_{ik} > \Theta_k$ at a certain time does not necessarily mean that the resource utilization of this PM will continually exceed the threshold for a certain period of time in the future. Furthermore, Xen live migration is CPU intensive, which may degrade the performance of both the source and destination PMs. Without sufficient resource, a VM migration will take a long time to finish, which will increase the service latency of tasks running on the PMs and may result in SLO violations. Therefore, we need to avoid triggering unnecessary migrations.

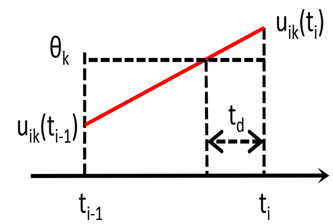


Fig. 5: Duration of overload status.

For this purpose, we specify that a migration is triggered only if the overload status of the PM (i.e., $u_{ik} > \Theta_k$) will last continuously for at least T_{t_d} time units, where T_{t_d} is the duration of overload status of the PM from time t_{i-1} to time t_i. The value of T_{t_d} determines the balance between offloading the overload resource and avoiding migrations due to transient overload status, and it can be tuned by the cloud provider. We can either set T_{t_d} to a constant time or make T_{t_d} a function of the VM migration time based on the requirement of guaranteeing SLO. We will first demonstrate when to trigger VM migration based on T_{t_d} and then discuss how to determine T_{t_d} so that the number of migrations is minimized without increasing the number of SLO violations.

Suppose the monitoring interval is Δt time units. That is, a PM checks its resource utilization every Δt time units, i.e., $\Delta t = t_i - t_{i-1}$, where t_i is current time and t_{i-1} is the time of last monitoring. As in [27], we assume that the resource utilization linearly increases from $u_{ik}(t_{i-1})$ to $u_{ik}(t_i)$ during time interval $\Delta\ t$. As shown in Figure 5, suppose a PM detects that its resource utilization exceeds the threshold ($u_{ik}(t_i) > \Theta_k$) at time t_i. According to historical record, it has resource utilization $u_{ik}(t_{i-1})$ at time t_{i-1}. The duration of overload status of the PM, t_d, can be calculated by

$$t_d = \frac{u_{ik}(t_i) - \Theta_k}{u_{ik}(t_i) - u_{ik}(t_{i-1})}\Delta t \qquad (37)$$

Then, VM migration will be triggered if and only if $t_d > T_{t_d}$.

We then discuss how to determine the value of T_{t_d} with the consideration of the SLO requirement. In this paper, we define SLO as the requirement that ε (in percentage) of resource demands of a VM must be satisfied during its lifetime [28]. We use t_s to denote the start time of a VM, and use t_v to denote the cumulated time that this VM has experienced resource overload since t_s until last monitoring time t_{i-1}. Considering that the time to complete VM migration is $\frac{M_{ij}}{B_{ip}}$,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing

the total time that the VM will experience overload before migration is completed equals $T_{t_d} + \frac{M_{ij}}{B_{ip}} + t_v$. We delay the migration as much as possible by fully taking advantage of SLO that allows $1 - \varepsilon$ violations. That is, if current time t_i is the migration start time of a VM in the PM, the VM should satisfy:

$$\frac{T_{t_d} + \frac{M_{ij}}{B_{ip}} + t_v}{t_i - t_s + \frac{M_{ij}}{B_{ip}}} = \varepsilon \tag{38}$$

Finally, we can get $T_{t_d} = \varepsilon(t_i - t_s + \frac{M_{ij}}{B_{ip}}) - \frac{M_{ij}}{B_{ip}} - t_v$. Therefore, in order to determine T_{t_d}, we need to record the start time t_s of each VM in the PM, and have the variable t_v to keep track of the cumulated SLO violation time of each VM.

4.6 Decentralized Destination PM Selection

Recall that the VM selection is conducted in each PM in a distributed manner, but the destination PM is selected in a central server because it needs to be chosen from all PMs. The centralized approach for destination PM selection is not efficient for a large scale cloud datacenter, because the amount of information required for this algorithm increases and may overburden the centralized server. In order to relieve the load of the centralized server, we develop a distributed version of the PM selection algorithm. The topology of a cloud datacenter can be abstracted by a graph with its nodes indicating PMs and switches and edges indicating physical links that connect PMs and switches. In this paper, we focus on tree-like topologies [29], [30], which are typical topologies of today's datacenters. As shown in Figure 6, we partition all the nodes in cloud datacenter into small clusters and each cluster consists of the nodes in one rack. Then, the load balancing is conducted within each cluster. That is, the VMs are migrated between physically close nodes. This way, the performance degradation due to VM migration can be reduced.

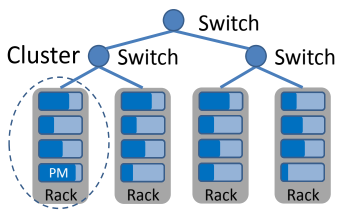
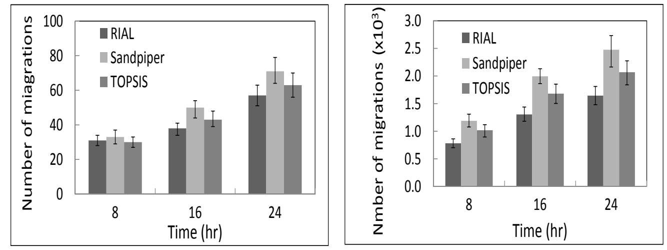


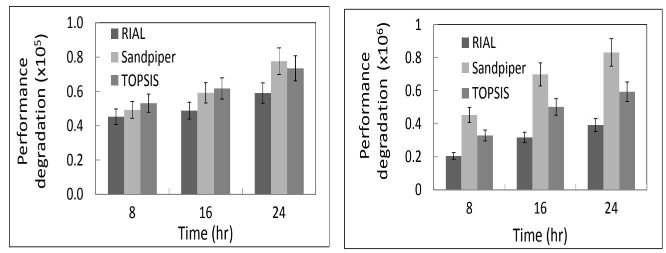
Fig. 6: Datacenter network.

Within each cluster, the nodes select the cluster master, who is responsible for selecting PMs for VM migrations in this cluster. This selected PM should not be overutilized and at the same time has the least probability to be selected as the destination PM, that is, it has the highest resource utilization. Unlike the centralized algorithm, in which a centralized server collects the information required in Equ. (22) from all the PMs in the datacenter, in the decentralized algorithm, the information is sent from every PM to its cluster master. For example, every PM in the cluster reports its status (i.e., resource utilization, communication rate with other PMs in this cluster) periodically (i.e., 5 minutes). The VM selection is conducted distributively in each PM. When a PM detects that it is overloaded, it selects its migration VMs and submits VM migration requests to its cluster master. Upon receiving the VM migration requests from the PMs, the cluster master then determines the ideal destination PM in its cluster, and selects PMs for the migration VMs based on Equ. (22). By limiting the PM selection within a small cluster (as opposed to the whole datacenter), we can increase the scalability of the PM selection algorithm. We will compare the the distributed algorithm and the centralized algorithm in Section 5.



(a) 100 PMs and 250 VMs (b) 1000 PMs and 5000 VMs

Fig. 7: Total number of VM migrations.

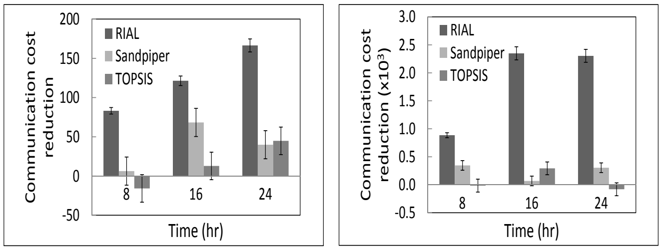


(a) 100 PMs and 250 VMs (b) 1000 PMs and 5000 VMs

Fig. 8: Total VM performance degradation.

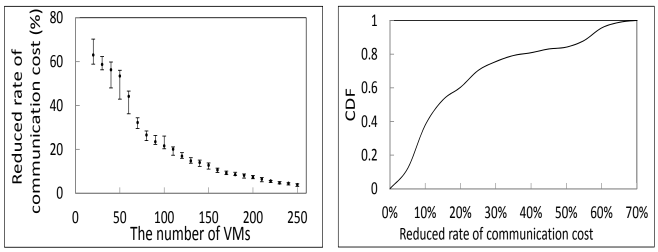
5 Performance Evaluation

We used the CloudSim [27] simulator and our deployed small-scale real-world testbed to evaluate the performance of RIAL in comparison to Sandpiper [1] and TOPSIS [5]. We used the real workload trace available in CloudSim to generate each VM's CPU resource consumption [23], [31]. To simulate memory and bandwidth usage, as in [19], we generated 5 different groups of (mean, variance range) for resource utilization, (0.2,0.05),(0.2,0.15),(0.3,0.05),(0.6,0.10),(0.6,0.15), and set each VM's memory/bandwidth utilization to a value generated by a randomly chosen group. Each PM has 1GHz 2-core CPU, 1536MB memory, and 1GB/s network bandwidth. Each VM has 500Hz CPU, 512MB memory, and 100Mbit/s bandwidth. With our experiment settings, the bandwidth consumption will not overload PMs due to their high network bandwidth. In CloudSim, we conducted experiments for two cloud scales. In the small scale experiment, we simulated 250 VMs running on 100 PMs. In the large scale experiment, we simulated 5000 VMs running on 1000 PMs. We generated a tree-like topology to connect the PMs, and measured the transmission delay between PMs based on the number of switches between them [20]. At the beginning of experiments, we randomly and evenly mapped the VMs to PMs. The overload threshold was set to 0.75. The weights for different resource are the same for Sandpiper or set to predefined ratio (e.g., 9:4 for CPU:MEM) as adopted in their papers. The load balancing algorithm was executed every 5 minutes. As in [19], we generated a random graph $G(n, p = 0.3)$ to simulate the VM communication topology, where n is the number of VMs and p is the probability that a VM communicates with another VM. The weight of each edge was randomly selected from [0,1] to represent the communication rate between two VMs. Unless otherwise specified, we repeated each test 20 times with a 24 hour trace and recorded the median, the 90th and 10th percentiles of the results.



(a) 100 PMs and 250 VMs (b) 1000 PMs and 5000 VMs

Fig. 9: Total VM communication cost reduction.



(a) Reduced rate of communication cost (b) CDF over all experiments

Fig. 10: Communication cost reduction of RIAL over Sandpiper/TOPSIS.

5.1 The Number of Migrations

Figure 7(a) and Figure 7(b) show the median, 10th percentile and 90th percentile of the total number of VM migrations by the time $t = 8h, 16h, 24h$ of the three methods in the small-scale and large-scale tests, respectively. We see that RIAL generates fewer migrations than Sandpiper and TOPSIS. Since RIAL considers resource intensity of different resources, it migrates fewer VMs from a PM to relieve its extra load. Also, RIAL proactively avoids overloading the destination PMs in the future. Thus, it keeps the system in a balanced state for a relatively longer period of time, resulting in fewer VM migrations than Sandpiper and TOPSIS within the same period of time. We also see that TOPSIS produces fewer VM migrations than Sandpiper because TOPSIS gives different weights to different resources while Sandpiper treats different resource equally. Additionally, we see that the three methods exhibit similar variances due to the initial random VM assignment to PMs.

5.2 VM Performance Degradation due to Migrations

We measured the total performance degradation of all migration VMs based on Equ. (7). Figure 8(a) and Figure 8(b) show the median, 90th and 10th percentiles of the total performance degradation (Formula (7)) in the small-scale and large-scale tests, respectively. We see that the total performance degradation of RIAL is lower than those of TOPSIS and Sandpiper in both small and large scale tests. This is caused by the distinguishing features of RIAL. First, RIAL triggers fewer VM migrations. Second, RIAL tries to minimize performance degradation in destination PM selection. Third, RIAL chooses VMs with lower utilizations of the non-intensive resources. TOPSIS generates lower performance degradation than Sandpiper because it generates fewer VM migrations as shown in Figure 7. We also see that in both the small-scale and large-scale tests, the performance degradation variance of the three methods follows RIAL<TOPSIS<Sandpiper though the difference is small in the small-scale test.

5.3 VM Communication Cost Reduction

The communication cost between a pair of VMs was measured by the product of their communication rate and transmission delay. We calculated the *communication cost reduction* by subtracting the total communication cost observed at a certain time point from the initial total communication cost of all VMs. Figure 9(a) and Figure 9(b) show the median, the 90th and 10th percentiles of total communication cost reduction at different time points in the small-scale and large-scale tests, respectively. We see that RIAL's migrations reduce much more communication cost than TOPSIS and Sandpiper, which may even increas the communication cost by migrations (shown by the negative results). RIAL exhibits smaller variance because RIAL tries to reduce VM communication rate between PMs caused by VM migration, while the other two methods do not consider it.

We then directly compare the communication costs after the migrations between different methods. We measured the communication costs of RIAL (x) and Sandpiper/TOPSIS (y) at the end of simulation and calculated the *reduced rate of communication cost* by $(y - x)/y$. We varied the number of VMs from 20 to 250 with an increment of 10, and mapped the VMs to 50 PMs. Each experiment is run for 30 times. As the reduced rates of RIAL over Sandpiper and TOPSIS are similar, we only show one result to make the figures clear.

Figure 10(a) shows the median, 10th percentile and 90th percentile of the reduced rate of communication cost with different numbers of VMs. We see that a smaller number of VMs lead to higher reduced rate of communication cost, which implies that RIAL can reduce more communication cost with fewer VMs relative to PMs. This is due to the fact that fewer VMs lead to fewer overloaded PMs hence more PM choices for a VM migration, which helps RIAL reduce more communication costs. Figure 10(b) plots the cumulative distribution function (CDF) of all 30*24 experiments versus the reduced rate of communication cost. We see that RIAL consistently outperforms Sandpiper and TOPSIS with lower communication cost in all experiments, and decreases the communication cost by up to 70%.

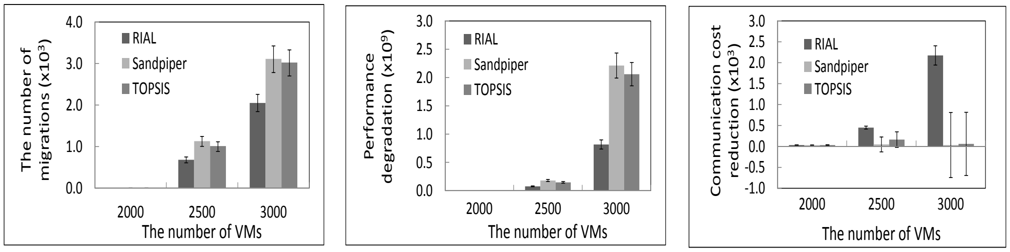
5.4 Performance of Varying Number of VMs and PMs

We then study the impact of different ratios of the number of VMs to the number of PMs on performance. Accordingly, we conducted two sets of tests. One test has 500 PMs with the number of VM varying from 2000 to 3000, and the other test has 1000 PMs with the number of VM varying from 4000 to 6000.

Figure 11(a) and Figure 12(a) show the median, 10th percentile and 90th percentile of the total number of migrations in the two tests, respectively. As the number of VMs increases, the total load on the cloud increases, resulting in more overloaded PMs and hence more VM migrations. When the number of VMs is 1000, the resource requests by VMs in the cloud is not intensive and only a few migrations are needed. When there are more VMs, the result of number of VM migrations follows RIAL<TOPSIS<Sandpiper, which is consistent with Figure 7 due to the same reasons.

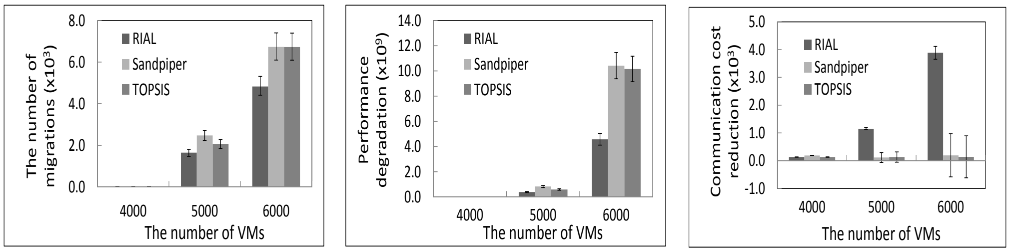
Figure 11(b) and Figure 12(b) show the results of the total VM performance degradation in the two tests, respectively. As the number of VM increases, the performance degradation increases in each method, mainly because of more triggered VM migrations. RIAL generates

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing



(a) The number of VM migrations (b) Performance degradation (c) Communication cost reduction

Fig. 11: Performance with varying VM to PM ratio (500 PMs).



(a) The number of VM migrations (b) Performance degradation (c) Communication cost reduction

Fig. 12: Performance with varying VM to PM ratio (1000 PMs).

lower performance degradation than Sandpiper and TOPSIS, especially with a higher number of VMs. We also see that the relative performance on the median, 10th percentile and 90th percentile between the three methods is aligned with that in Figure 8 due to the same reasons.

Figure 11(c) and Figure 12(c) show the results of the total communication cost reduction in the two tests, respectively. When the VM number is small, there is only a few VM migrations, resulting in small cost reduction and small variance for all methods. As the number of VMs grows, RIAL achieves a higher cost reduction than Sandpiper and TOPSIS. Also, RIAL has much smaller variance than Sandpiper and TOPSIS as the error bars indicate. Both Sandpiper and TOPSIS performs similarly since neither of them considers the VM communications when selecting VMs and PMs. The relative performance between the three methods is consistent with that in Figure 9 due to the same reasons.

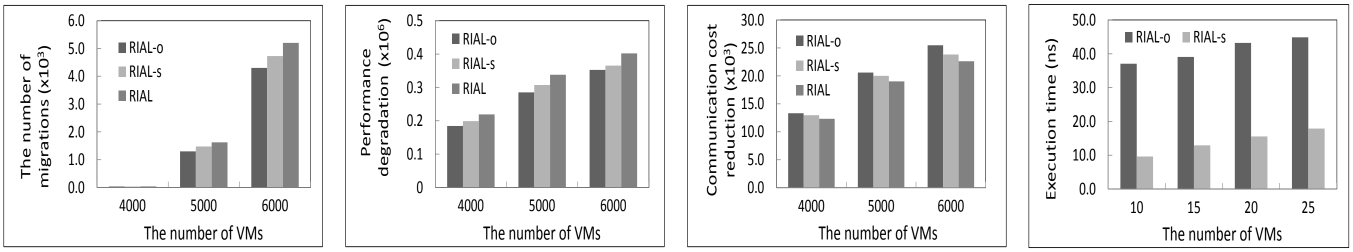
Comparing Figure 11 and Figure 12, we see that the results in Figure 12 have higher absolute values than those in Figure 11 because the workload and the scale of the cloud are doubled. We can conclude from 11 and Figure 12 that RIAL outperforms Sandpiper and TOPSIS under varying ratios of the number of VMs to PMs in terms of the number of VM migrations, VM performance degradation and communication cost.

5.5 Performance of Weight Determination Algorithms

We study the performance of different weight determination algorithms introduced in Section 4.3. In the following sections, we adopt the same setting for the large scale (1000 PMs) and vary the number of VMs from 4000 to 6000 with an increment of 2000 at each step, unless otherwise specified. We use RIAL-o to denote the optimal weight determination algorithm based on Equ. (28) and Equ. (36), use RIAL-s to denote the simplified algorithm based on Equ. (31), and use RIAL to denote the algorithm with constant weights (i.e., $w_t = 3$, $w_d = 3$). Figure 13(a) shows the number of VM migrations, which follows RIAL-o<RIAL-s<RIAL. This is because RIAL-o guarantees that the weights of overutilized resource are not overtaken by the weights for communication rate and performance degradation but RIAL-s and RIAL cannot. Therefore, RIAL-o needs fewer migrations to offload extra load of overutilized resources. RIAL-s outperforms RIAL because RIAL-s determines w_t and w_d based on the weights for resources while RIAL uses constant w_t and w_d, which may make w_t and w_d overtake the resource weights. The number of migrations increases with the number of VMs because more VMs imposes more workload on the same number of PMs (i.e., 1000 PMs). Figure 13(b) shows the performance degradation, which follows RIAL-o<RIAL-s<RIAL because fewer migrations lead to less performance degradation. The performance degradation increases with the number of VMs due to the same reason as Figure 13(a). Figure 13(c) shows the communication cost reduction, which follows RIAL-o>RIAL-s>RIAL. This is because the amount of sacrifice on the priority of offloading the excess load to reducing communication cost follows RIAL-o<RIAL-s<RIAL in weight determination. The communication cost reduction increases with the number of VMs due to the same reason mentioned before. We also measured the execution time of the wight determination algorithms by varying the number of VMs in a PM from 10 to 25 with an increment of 5 at each step. Figure 13(d) shows the execution time of the different algorithms with different number of VMs in a PM. We see that RIAL-s is faster than RIAL-o due to the simpleness of Equ. (31) compared to Equ. (28). We do not present RIAL here because it has zero execution time (constant complexity). This result confirms the feasibility of RIAL-s as it can achieve similar performance as RIAL-o while consuming less time.

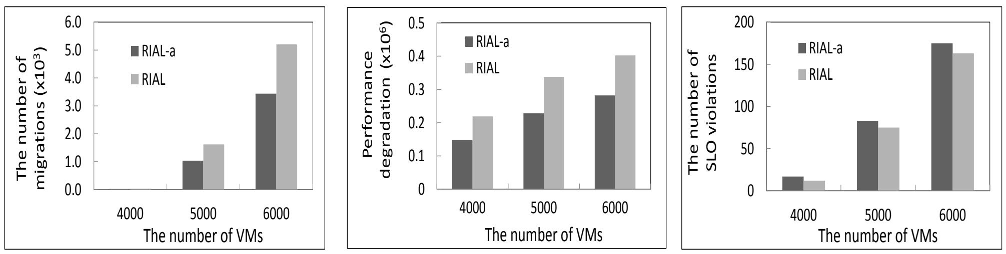
5.6 Performance of Migration Triggering Algorithm

We use RIAL-a to denote RIAL that avoids unnecessary migrations using the migration triggering policy. We set $\varepsilon = 0.95$ and determine T_{t_d} based on Equ. (38). The number of SLO violations is the number of VMs that have experienced overload status for a duration more than 1-ε percent of their lifetimes. Figure 14(a) shows the number of VM migrations. We see that RIAL-a triggers a fewer



(a) The number of VM migrations (b) Performance degradation (c) Communication cost reduction (d) Execution time

Fig. 13: Performance of weight determination algorithms with varying VM to PM ratio (1000 PMs).



(a) The number of VM migrations. (b) Performance degradation. (c) The number of SLO violations.

Fig. 14: Performance of migration triggering algorithm.

number of VM migrations than RIAL since it avoids unnecessary VM migrations and meanwhile avoids violating SLO requirements. The number of VM migrations increases as the number of VMs increases since more VMs aggravate the load in the datacenter. Figure 14(b) shows the total performance degradation. We see that the total performance degradation of RIAL-a is lower than RIAL. This is mainly because that RIAL-a avoids unnecessary VM migrations and triggers fewer VM migrations. The performance degradation increases with the number of VMs since move VMs generate more workload and hence more VM migrations. Figure 14(c) shows the number of SLO violations. We see that RIAL-a produces a similar number of SLO violations as RIAL although RIAL-a does not immediately trigger VM migration upon detecting $u_{ik} > \Theta_k$. Also, the number of SLO violations increases with the number of VMs due to higher workloads on PMs. This result confirms that triggering VM migration only when the overload status of a PM lasts continuously for at least T_{t_d} time can improve the performance of RIAL without significantly affecting SLO.

5.7 Performance of Decentralized Destination PM Selection

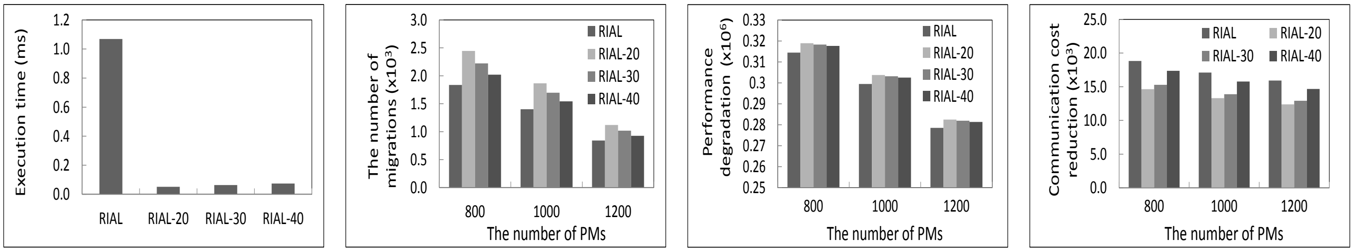
We then compare the performance of decentralized destination PM selection algorithm introduced in Section 4.6 with the centralized algorithm. We denote the centralized algorithm as RIAL, and the decentralized algorithms with cluster size c as RIAL-c, where c was set to 20, 30 and 40, respectively. Figure 15(a) shows the execution time of different algorithms. We see that the execution time follows RIAL-20<RIAL-30<RIAL-40<RIAL. This is because a cluster with a smaller number of PMs has a smaller problem size and all cluster masters conduct the destination PM selection simultaneously. RIAL has a higher time than the others because it must rank all the PMs based on Equ. (22) in the datacenter for PM selection. Figure 15(b) shows the number of VM migrations, which follows RIAL<RIAL-40<RIAL-30<RIAL-20. This is because the selected destination PM in a smaller cluster is not the ideal destination PM in the system scope with high probability and is more likely to become overloaded later on, which leads to more VM migrations. Figure 15(c) shows the performance degradation, which follows RIAL<RIAL-40<RIAL-30<RIAL-20. Although selecting PM nearby (in a smaller cluster) can reduce the distance of migration, but the large number of VM migrations (as indicated in Figure 15(b)) offsets the benefit, resulting in higher performance degradation. Figure 15(d) shows the communication cost reduction, which follows RIAL-20<RIAL-30<RIAL-40<RIAL due to the reason that a larger cluster has more opportunities or options for reducing communication cost. The best PM selected within a cluster reduces less communication cost compared to the best PM selected within the whole datacenter. These results confirms that the decentralized algorithm does not degrade the performance greatly while significantly reduces the algorithm execution time.

5.8 Real-World Testbed Experiments

For real-world testbed experiments of RIAL, we deployed a cluster with 7 PMs (2.00GHz Intel(R) Core(TM)2 CPU, 2GB memory, 60GB HDD) and two NFS (Network File System) servers with a combined capacity of 80GB. We then implemented the various load balancing algorithms in Python 2.7.2 using the XenAPI library [32] running in a management node (3.00GHz Intel(R) Core(TM)2 CPU, 4GB memory, running Ubuntu 11.04). We created 15 VMs (1VCPU, 256MB memory, 8.0GB virtual disk, running Debian Squeeze 6.0) in the cluster; each with Apache2 Web Server installed. We used the publicly available workload generator *lookbusy* [33] to generate both CPU and memory workloads. We recorded the generated CPU and memory workloads. The actually provisioned resources can be collected from the management node. The load balancing was executed once every 5 minutes. Similar to the simulation experiment, we set the overload threshold $T_{h,k}$ to 0.75. Initially, we randomly assigned the VMs to PMs, and then compared the performance of different load balancing strategies.

The communication delay between two PMs is determined by the number of switches across the communication paths in the testbed architecture. We

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing



(a) Execution time (b) The number of VM migrations (c) Performance degradation (d) Communication cost reduction

Fig. 15: Performance of decentralized destination PM selection algorithm with varying VM to PM ratio (1000 PMs).

created latency between machines such that all traffic from machine is in the ratio of 1:4:10 to follow the network hierarchical setup [34]. That is, if the communication path between two PMs comes across one switch, two switches, and three switches, respectively, the latency between VMs in the two PMs was set to be 1, 4 and 10, respectively. We run each test for 20 times; each lasts for approximately 60m.

5.8.1 The Number of Migrations

Figure 16 shows the median, 10th percentile and 90th percentile of the total number of migrations in different methods. We can see that RIAL triggers fewer VM migrations than the other two methods to achieve a load balanced state, while TOPSIS generates fewer VM migrations than Sandpiper. Figure 17 shows the accumulated number of migrations over time. We see that before 40m, RIAL generates a similar number of migrations as Sandpiper and TOPSIS, since all methods begin from a similar load unbalanced state at the beginning of the experiment. After around 40m, RIAL produces much fewer migrations and after 50m, it produces no migrations and reaches the load balanced state, while TOPSIS and Sandpiper continue to trigger VM migrations. This result confirms that RIAL generates fewer migrations and achieves the load balanced state faster due to its consideration of resource intensity.

5.8.2 Performance Degradation

Figure 18 shows the median, 10th percentile and 90th percentile of the total VM performance degradation of the three methods. We measured the real migration time to replace $\frac{M_{ij}}{B_{ip}}$ in Formula (7) to calculate the performance degradation. The figure shows that the VM performance degradation of RIAL is lower than those of Sandpiper and TOPSIS since it tries to reduce VM performance degradation when selecting destination PMs. TOPSIS has a slightly lower VM performance degradation than Sandpiper. As in the simulation, the variance of the results also follows RIAL<TOPSIS<Sandpiper though it is not obvious due to the small scale. These experimental results confirm the advantage of RIAL with the consideration of VM performance degradation in load balancing.

5.8.3 Communication Cost

We generated a random graph $G(n = 15, p = 0.2)$ to represent the VM communication topology. Initially, we manually placed intensively communicating VMs in PMs with higher network delay for testing.

We measured the sum of the communication cost of each pair of communicating VMs at the initial stage as the base and measured the total communication cost at every 5 minutes during the experiment. Figure 19 shows the normalized communication cost according to the base. We see that as time goes on, the communication cost of all methods decreases. This is because we initially placed intensively communicating VMs in PMs with higher network delay and VM migration can reduce the communication cost. Our method can reduce the communication cost much more and faster than the other methods, reaching 20% of the base communication cost. TOPSIS and Sandpiper have similar curves since they neglect VM communication cost in load balancing.

6 Conclusions

In this paper, we propose a Resource Intensity Aware Load balancing (RIAL) method in clouds that migrates VMs from overloaded PMs to lightly loaded PMs. It is distinguished by its resource weight determination based on resource intensity. In a PM, a higher-intensive resource is assigned a higher weight and vice versa. By considering the weights when selecting VMs to migrate out and selecting destination PMs, RIAL achieves faster and lower-cost convergence to the load balanced state, and reduces the probability of future load imbalance. Further, RIAL takes into account the communication dependencies between VMs in order to reduce the communication between VMs after migration, and also tries to minimize the VM performance degradation when selecting destination PMs. The weights assigned to communication cost and performance degradation are optimally determined so that the overutilized resource is relieved and both communication cost and performance degradation are minimized. We also propose RIAL with a more strict migration triggering algorithm to avoid unnecessary migrations while satisfying SLOs. Finally, we make RIAL decentralized to improve its scalability. Both trace-driven simulation and real-testbed experiments show that RIAL outperforms other load balancing approaches in regards to the number of VM migrations, VM performance degradation and VM communication cost. In our future work, we will study how to globally map migration VMs and destination PMs in the system to enhance the effectiveness and efficiency of load balancing. We will also measure the overhead of RIAL and explore methods to achieve an optimal tradeoff between overhead and effectiveness.

Acknowledgements

This research was supported in part by U.S. NSF grants ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751. We would like to thank Dr. Liuhua Chen for his valuable contribution. An early version of this work was presented in the Proceedings of Infocom 2014 [35].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2737628, IEEE Transactions on Cloud Computing

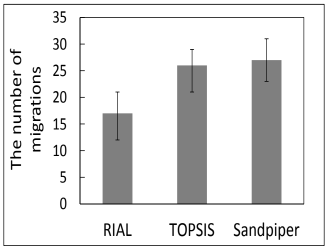


Fig. 16: Number of migrations.

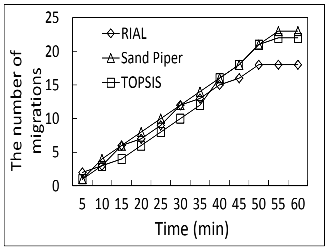


Fig. 17: Accumulated # of migrations.

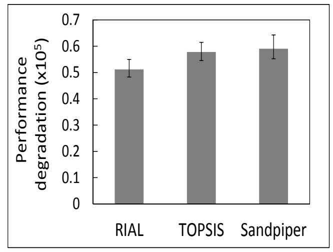


Fig. 18: Performance degradation.

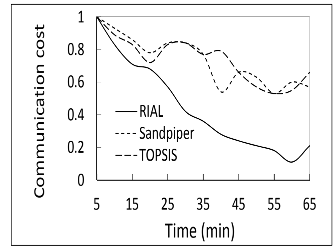


Fig. 19: Communication cost.

REFERENCES

[1] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Black-box and gray-box strategies for virtual machine migration." in *Proc. of NSDI*, vol. 7, 2007, pp. 17–17.

[2] E. Arzuaga and D. R. Kaeli, "Quantifying load imbalance on virtualized enterprise servers." in *Proc. of WOSP/SIPEW*, 2010, pp. 235–242.

[3] A. Singh, M. R. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers." in *Proc. of SC*, 2008, p. 53.

[4] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *Proc. of NOMS*, 2006, pp. 373–381.

[5] M. Tarighi, S. A. Motamedi, and S. Sharifian, "A new model for virtual machine migration in virtualized cluster server based on fuzzy decision making." *arXiv preprint arXiv:1002.3329*, 2010.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines." in *Proc. of NSDI*, 2005, pp. 273–286.

[7] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for hpc with xen virtualization." in *Proc. of ICS*, 2007, pp. 23–32.

[8] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers." in *Proc. of WWW*, 2007.

[9] F. Xu, F. Liu, and H. Jin, "Managing performance overhead of virtual machines in cloud computing: A survey, state of art and future directions," in *Proc. of IEEE*, 2014.

[10] X. Li, J. Wu, S. Tang, and S. Lu, "Let's stay together: Towards traffic aware virtual machine placement in data centers," in *Proc. of INFOCOM*, 2014.

[11] F. Xu, F. Liu, and H. Jin, "Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud," *IEEE Transactions on Computers*, 2016.

[12] S. Lim, J. Huh, Y. Kim, and C. R. Das, "Migration, assignment, and scheduling of jobs in virtualized environment," in *Proc. of HotCloud*, 2011.

[13] H. Hsiao, H. Su, H. Shen, and Y. Chao, "Load rebalancing for distributed file systems in clouds." *TPDS*, vol. 24, no. 5, pp. 951–962, 2012.

[14] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks." in *Proc. of SIGCOMM*, vol. 41, no. 4, 2011, pp. 242–253.

[15] D. Xie, N. Ding, Y. C. Hu, and R. R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers." in *Proc. of SIGCOMM*, vol. 42, no. 4, 2012, pp. 199–210.

[16] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing." in *Proc. of SIGCOMM*, 2012, pp. 187–198.

[17] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers." in *Proc. of INFOCOM*, 2011, pp. 1098–1106.

[18] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters." in *Proc. of INFOCOM*, 2012, pp. 702–710.

[19] V. Shrivastava, P. Zerfos, K. Lee, H. Jamjoom, Y. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers." in *Proc. INFOCOM*, 2011, pp. 66–70.

[20] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement." in *Proc. of INFOCOM*, 2010, pp. 1154–1162.

[21] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems." in *Proc. of SOCC*, 2011, p. 5.

[22] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation." *CoRR*, vol. abs/1109.4974, 2011.

[23] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers." *CCPE*, vol. 24, pp. 1397–1420, 2012.

[24] M. J. Magazine and M.-S. Chern, "A note on approximation schemes for multidimensional knapsack problems." *MOR*, 1984.

[25] H. Wang and C. L. Yoon, "Multiple attributes decision making methods and applications." *Springer*, 1981.

[26] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proc. of EuroSys*, 2015, p. 18.

[27] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." *SPE*, vol. 41, pp. 23–50, 2011.

[28] A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica, "Cake: enabling high-level SLOs on shared storage systems," in *Proc. of SOCC*, 2012, p. 14.

[29] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *Proc. of SIGCOMM*, 2009.

[30] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, Aug. 2008.

[31] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab." *OSR*, vol. 40, pp. 65–74, 2006.

[32] "Xenapi," http://community.citrix.com/display/xs/Download+SDKs, [accessed in Feb. 2015].

[33] "lookbusy," http://devin.com/lookbusy/, [accessed in Feb. 2015].

[34] C. Peng, M. Kim, Z. Zhang, and H. Lei, "VDN: Virtual machine image distribution network for cloud data centers." in *Proc. of INFOCOM*, 2012, pp. 181–189.

[35] L. Chen, H. Shen, and K. Sapra, "RIAL: Resource intensity aware load balancing in clouds." in *Proc. of INFOCOM*, 2014.

Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the Department of Electrical and Computer Engineering at Clemson University. Her research interests include distributed computer systems and computer networks, with an emphasis on peer-to-peer and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a Microsoft Faculty Fellow of 2010, a senior member of the IEEE and a member of the ACM.