RIAL: Resource Intensity Aware Load Balancing in Clouds

Liuhua Chen, Haiying Shen, Karan Sapra
Department of Electrical and Computer Engineering
Clemson University, Clemson, South Carolina 29634
Email: {liuhuac,shenh,ksapra}@clemson.edu

***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 heterogenous 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 leads 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 migrate VMs to PMs with minimum VM performance degradation. Our extensive trace-driven simulation results and real-world experimental results show the superior performance of RIAL compared to other load balancing methods.**

I. Introduction

Cloud computing is becoming increasingly popular due to its ability to provide unlimited computing services with the pay-as-you-go service 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.

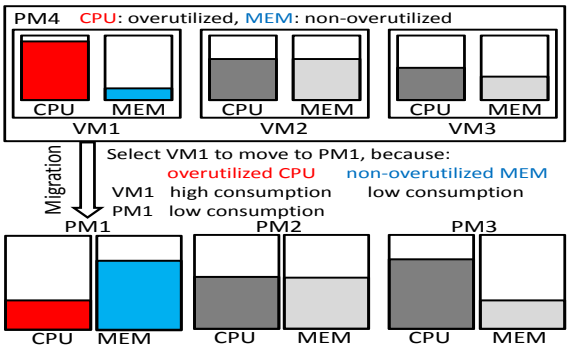


Fig. 1. Migration VM and destination PM selection policy.

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.

Also, previous load balancing methods do not consider the communication between VMs and VM performance (i.e., response time) degradation due to migration. There may be intensive network communication between two VMs, so

978-1-4799-3360-0/14/$31.00

separating such two VMs to two different PMs would increase the network bandwidth consumption. Moving a VM to a distant PM would lead to high VM performance degradation. Therefore, the previous methods are not efficient for cloud tasks where VM communication is intensive and delayed VM response time is highly undesirable.

We aim to not only reduce the number of VM migrations in achieving the load balanced state but also avoid load imbalance in the future (i.e., fast and constant convergence) while minimize the adverse effect of VM 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 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. Consider 4 PMs in Figure 1, where overloaded PM4 hosts 3 VMs. Because CPU is overutilized while MEM is underutilized in PM4, 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. 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. 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 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 II briefly describes the related work. Section III presents the objective of RIAL. Section IV presents the detailed design of RIAL and an analysis of its performance compared to other load balancing methods. Section V evaluates RIAL in both simulation and real-world experiments in comparison with other load balancing methods. Finally, Section VI summarizes the paper with remarks on our future work.

II. 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. 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.

Some works deal with load balancing on one resource such as storage [8] and bandwidth [9]–[11]. Hsiao *et al.* [8] proposed a load balancing algorithm for distributed file systems in clouds by moving file chunks from overloaded servers to lightly loaded servers. Oktopus [9] provides static reservations throughout the network to implement bandwidth guarantees. Popa *et al.* [11] navigated the tradeoff space of requirements-payment proportionality, resource minimum guarantee and system utilization when sharing cloud network bandwidth. Xie *et al.* [10] 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 [12]–[15]. Lin *et al.* [12] 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.* [13] focused on resource allocation that balances the load among servers to achieve throughput optimization. Meng *et al.* [15] used traffic patterns among VMs to determine VM placement in order to improve network scalability. Shrivastava *et al.* [14] 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. Shen *et al.* [16] proposed an online resource demand prediction method to achieve adaptive resource allocation.

III. Objectives and Problem Statement

A. 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 $T_{h,k}$ 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 $u_{ik} < T_{h,k}$ for each of its type-k resource (i.e., lightly loaded status). We call a PM with $u_{ik} > T_{h,k}$ *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

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

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 [8] 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.

B. 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.

C. Reducing VM Performance Degradation by Migrations

When a VM is being migrated to another PM, its performance (response time) is degraded [17]. 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 [17], [18]:

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

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 [11], [15].

D. 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 VM communications between PMs and the VM performance degradation. 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:

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

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

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

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

Our problem of VM migration is a variant of the multiple knapsack problem, which is NP-complete [19]. A simpler formulation of our problem has been shown to be NP-complete in [14], [15]. 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.

IV. 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 $u_{ik} \leq T_{h,k}$ ($k \in K$). Below, we present the methods for selecting VMs to migrate and for selecting destination PMs with the objectives listed in Section III-D.

A. 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.

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

As a result, for an overutilized resource $k \in O$, a higher utilization leads to a higher weight. For a non-overutilized resource $k \in L$, a higher utilization leads to a lower weight. The weight of resource k (w_{ik}) means the priority of migrating this resource out. Note that $w_{ik} > 1$ for a resource $k \in O$ always 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. Then, we create a $1 \times |K|$ matrix $W_i = (w_{i1}, \cdots, w_{ik}, \cdots, w_{i|K|})$ for P_i as its criteria weighted matrix.

Recall that u_{ijk} is the type-k resource utilization rate of VM V_{ij}. Using the MCDM (Multi-Criteria Decision Making) method [20], 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.

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 III-B, 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.

B. 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 III. 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:

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

where

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

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

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

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.

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

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.

C. 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.

TABLE I
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

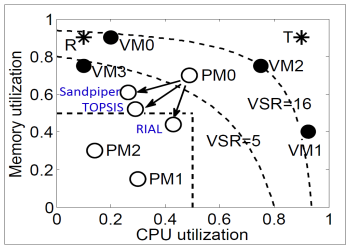
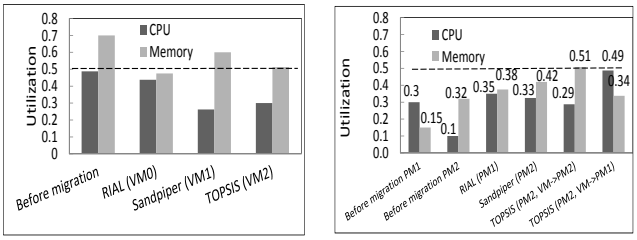


Fig. 2. VM and PM selection process.

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 2 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.



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

Fig. 3. Advantage of RIAL in reducing migrations.

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 2 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 2) 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 2)

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 3(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 2 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 VM migration. RIAL reduces both CPU and memory utilizations below the threshold.

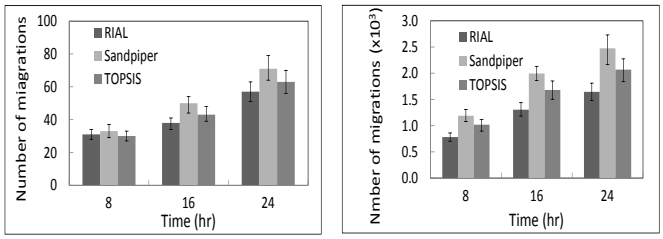
For destination PM selection, PM1(0.3,0.15) and PM2(0.1,0.32) are two candidates for the VM from PM0. Sandpiper selects the PM that has the least $volume$ as the destination, which is PM2. TOPSIS determines the ideal PM with the least CPU and memory utilization of all candidate PMs (i.e., (0.15, 0.1)), 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 3(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 I 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.

V. Performance Evaluation

We used the CloudSim [21] 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 [18], [22]. To simulate memory and bandwidth usage, as in [14], 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.



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

Fig. 4. Total number of VM migrations.

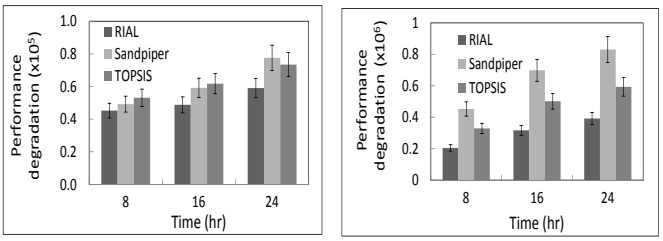
Each VM has 500Hz CPU, 512MB memory, and 100Mbit/s bandwidth. With our experiment settings, the bandwidth consumption will not overload PMs due to their high network bandwidth. In CloudSim, we conducted experiments for two cloud scales. In the small scale experiment, we simulated 250 VMs running on 100 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 [15]. 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 [14], 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. The Number of Migrations

Figure 4(a) and Figure 4(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.

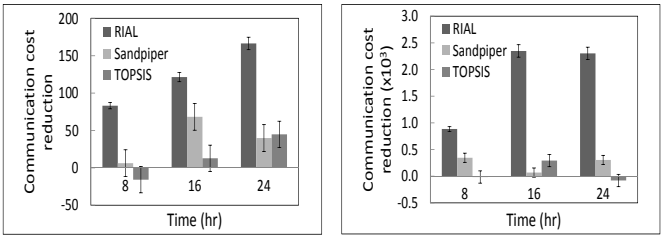
B. VM Performance Degradation due to Migrations

Figure 5(a) and Figure 5(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



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

Fig. 5. Total VM performance degradation.



(a) 100 PMs and 250 VMs (b) 500 PMs and 1250 VMs

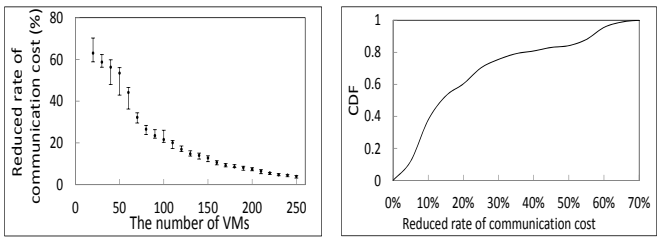
Fig. 6. Total VM communication cost reduction.

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 4. 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.

C. 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 6(a) and Figure 6(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



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

Fig. 7. Communication cost reduction of RIAL over Sandpiper/TOPSIS.

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 7(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 7(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%.

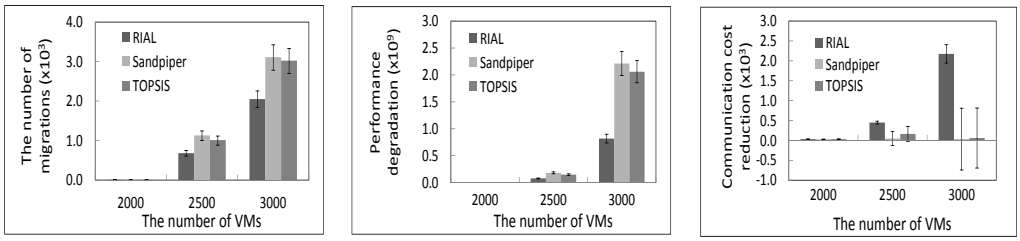
D. Performance of Varying Number of VMs and PMs

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 8(a) and Figure 9(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 4 due to the same reasons.

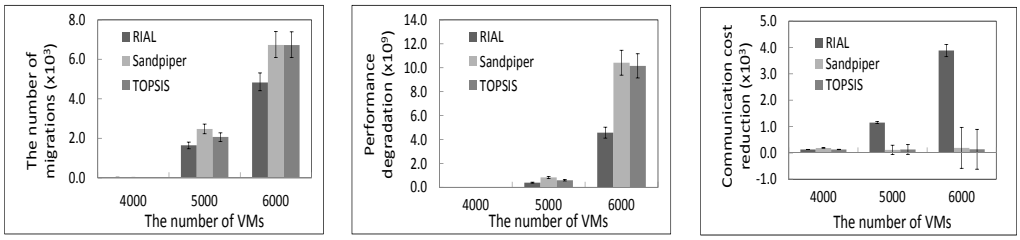
Figure 8(b) and Figure 9(b) show the results of the total VM performance degradation in the two tests, respectively. As the number of VM increases, the performance degradation increases in each method, mainly because of more triggered VM migrations. RIAL generates lower performance degradation than Sandpiper 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 5 due to the same reasons.

Figure 8(c) and Figure 9(c) show the results of the total communication cost reduction in the two tests, respectively.



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

Fig. 8. Performance with varying VM to PM ratio (500 PMs).



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

Fig. 9. Performance with varying VM to PM ratio (1000 PMs).

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 6 due to the same reasons.

Comparing Figure 8 and Figure 9, we see that the results in Figure 9 have higher absolute values than those in Figure 8 because the workload and the scale of the cloud are doubled. We can conclude from 8 and Figure 9 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.

E. 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 [23] 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* [24] to generate both CPU and memory workloads.

The communication delay between two PMs is determined by the number of switches across the communication paths in the testbed architecture. We created latency between machines such that all traffic from machine is in the ratio of 1:4:10 to follow the network hierarchical setup [25]. 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.

1) The Number of Migrations: Figure 10 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 11 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.

2) Performance Degradation: Figure 12 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.

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 commu-

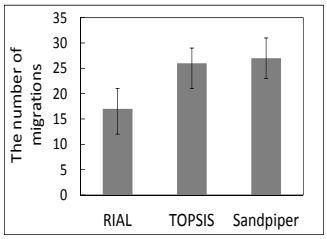


Fig. 10. Number of migrations.

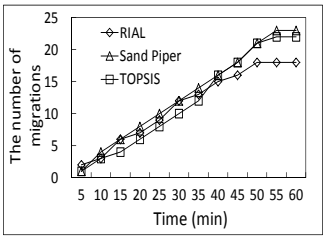


Fig. 11. Accumulated # of migrations.

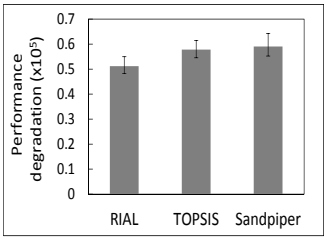


Fig. 12. Performance degradation.

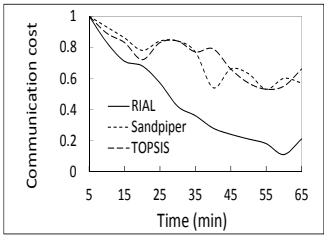


Fig. 13. Communication cost.

nicating 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 13 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.

VI. 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. 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 IIS-1354123, CNS-1254006, CNS-1249603, CNS-1049947, CNS-0917056 and CNS-1025652, Microsoft Research Faculty Fellowship 8300751.

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*, 2007.

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

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

[4] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments." in *Proc. of NOMS*, 2009.

[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." *CoRR*, 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.

[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.

[8] H. Hsiao, H. Su, H. Shen, and Y. Chao, "Load rebalancing for distributed file systems in clouds." *TPDS*, 2012.

[9] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks." in *Proc. of SIGCOMM*, 2011.

[10] 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*, 2012.

[11] 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.

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

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

[14] 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.

[15] 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.

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

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

[18] 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*, 2011.

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

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

[21] 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*, 2011.

[22] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab." *OSR*, 2006.

[23] "Xenapi," http://community.citrix.com/display/xs/Downl oad+SDKs.

[24] "lookbusy," http://devin.com/lookbusy/.

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