Stochastic Load Balancing for Virtual Resource Management in Datacenters

Lei Yu, Liuhua Chen, Zhipeng Cai, Haiying Shen, Yi Liang, Yi Pan

Abstract—Cloud computing offers a cost-effective and elastic computing paradigm that facilitates large scale data storage and analytics. By deploying virtualization technologies in the datacenter, cloud enables efficient resource management and isolation for various big data applications. Since the hotspots (*i.e.*, overloaded machines) can degrade the performance of these applications, virtual machine migration has been utilized to perform load balancing in the datacenters to eliminate hotspots and guarantee Service Level Agreements (SLAs). However, the previous load balancing schemes make migration decisions based on deterministic resource demand estimation and workload characterization, without considering their stochastic properties. By studying real world traces, we show that the resource demand and workload of virtual machines are highly dynamic and bursty, which can cause these schemes to make inefficient migrations for load balancing. To address this problem, in this paper we propose a stochastic load balancing scheme which aims to provide probabilistic guarantee against the resource overloading with virtual machine migration, while minimizing the total migration overhead. Our scheme effectively addresses the prediction of the distribution of resource demand and the multidimensional resource requirements with stochastic characterization. Moreover, as opposed to the previous works that measure the migration cost without considering the network topology, our scheme explicitly takes into account the distance between the source physical machine and the destination physical machine for a virtual machine migration. The trace-driven experiments show that our scheme outperforms the previous schemes in terms of SLA violation and the migration cost.

Index Terms—datacenter, virtual machine migration, load balance, stochastic load balancing, resource management.

1 Introduction

RECENTLY virtualization technologies have been widely deployed in data centers by the cloud providers to provide Infrastructure as a Service (IaaS), such as Amazon Elastic Compute Cloud (EC2) [3] and Microsoft Azure [1]. The virtual computation environment provides large scale on-demand and elastic computation and storage capabilities, which significantly facilitate large-scale data analytics and spur big data innovation. Through virtualization, the resources on Physical Machines (PMs) are partitioned into Virtual Machines (VMs), which host application computation and data while enabling application isolation from applications in other VMs. In the virtual machine environments, multiple VMs share the resources on the same physical machine. Each VM can run one or more applications and an application can distributedly run in multiple VMs.

Due to the dynamic workload of applications and the resource multiplex sharing of data center networks, guaranteeing the *Service Level Agreement* (SLA) of cloud applications, especially large-scale big data applications, is a complex task. The resource virtualization in the cloud facilitates such task. It enables elastic resource scaling that dynamically adjusts the resource allocation for a VM to accommodate the application resource demands [28]. It also enables virtual machine migration [36] for load balancing to eliminate hotspots and consolidation [40] to improve resource utilization and energy efficiency.

Load balancing is critical for guaranteeing the SLAs of applications in cloud. The workload increase of applications in the virtual machines may cause one or multiple resources including CPU, memory, I/O and network bandwidth on the physical machines overloaded. An overloaded physical machine often degrades the application performance of all the VMs on it, increasing the job completion time for batch data processing and the response time of interactive applications. In order to eliminate such hotspots, excess load must be migrated from the overloaded physical machines to underutilized ones. However, this load balancing through VM migration is a complicated and challenging problem. First, it needs to consider multiple resources (*e.g.*, CPU, memory, I/O and network) for each VM and physical machine. The applications have diverse resource demands, so the VMs running these applications can be CPU-intensive, memory-intensive or network-intensive. Thus, to decide which PM a VM should be migrated to, the multidimensional resource requirement of the VM has to be considered and matched with the multidimensional available resources on the PMs. Second, the overhead of VM migrations for load balancing (*i.e.*, the amount of data transferred) should be minimized. The VM migration can adversely affect the application performance in the VM [15], and incur severe performance interference to other running VMs on both migration source and destination PMs [38]. Reducing the overhead of VM migrations alleviates the performance degradation caused by VM migrations for load balancing. Third, due to the dynamic changes of application workload in VMs, it is not efficient to make the migration decision only based on the current state of the system. Accurate load prediction is

- *Lei Yu, Zhipeng Cai, Yi Liang and Yi Pan are with the Department of Computer Science, Georgia State University, Atlanta, GA, 30302. Z. Cai is the corresponding author of this paper. E-mail: {lyu13,yliang5}@student.gsu.edu, {zcai, yipan}@gsu.edu*
- *Liuhua Chen and Haiying Shen are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634. E-mail: {liuhuac, shenh}@clemson.edu*

Manuscript received April 19, 2005; revised September 17, 2014.

necessary for load balancing but difficult.

To address this load balancing problem, a number of methods [4], [6], [8], [10], [12], [18], [28], [30], [36] have been proposed which can be divided into two categories: reactive and proactive load balancing. The reactive methods [4], [12], [30], [36] determine load imbalance and hotspots by comparing the current resource utilization measurements with the given thresholds, and decide where the VM should be migrated based on the current load states of the PMs. The common issues of these reactive methods are the time delay to respond the load imbalances and inefficient load balancing actions due to the dynamic load changes. In order to address these disadvantages, proactive methods [6], [8], [10], [18], [28], [37] are proposed, which make VM migration decisions based on the predictions of resource utilizations of VMs and PMs. However, for highly dynamic workloads, the prediction-driven load balancing can be inefficient due to the inaccuracy of predicted resource demand and usage. Over-estimation predictions may cause wasteful resource allocation and under-estimation predictions can cause significant SLA violations. Previous work [28] uses adaptive padding to avoid under-estimation errors and fast correction after detecting under-estimation errors. Still, it makes VM migration decisions based on deterministic estimations of VM resource demands and current load states of PMs, without considering their stochastic variances. This may lead to inefficient load balancing for highly dynamic workloads, increasing the risk of SLA violations and the times of VM migrations.

To address demand uncertainty and dynamic workloads, in this paper we consider stochastic load balancing through VM migration. As opposed to the previous works, the stochastic load balancing problem characterizes the resource demand of VMs and load states of PMs probabilistically, and aims to ensure the aggregate utilization of each type of resources on each PM not exceeding its capacity with a high probability $1 - \epsilon$. The probability ϵ is determined by the SLA agreement and indicates the risk of SLA violations on each PM. The stochastic workload characterization is able to capture the uncertainty and dynamic changes of resource utilizations. With the probabilistic guarantee for handling overloads, the load balancing decision can ensure the resulted application performance is more resilient against highly dynamic workloads while achieving efficient statistical multiplexing of resources. However, the stochastic load balancing poses new challenging problems including how to estimate stochastic resource demand, how to detect hotspots and how to make VM migrations while capturing multidimensional stochastic resource requirements. Although the existing work [8] also aims to provide the same probabilistic guarantee, it just regards the demand prediction error as stochastic variables and does not fully address these problems in the context of stochastic demands. Its VM migration algorithm considers each resource separately, without combining all the dimensions to evaluate the overall load status of PMs. Furthermore, it does not consider the migration cost. In contrast, our paper proposes a stochastic load balancing scheme which effectively and efficiently addresses these problems.

Our scheme aims to minimize the transmission overhead incurred by VM migration. Previous methods [28], [36] decide VM migrations with the consideration of VMs' memory footprint. However, they measure the migration cost without considering the network topology. In contrast, the VM migration algorithm in our scheme takes into account the transmission distance (hops) for the migration cost. Furthermore, compared with the previous methods, our migration algorithm is able to improve the worst performance the system could experience from the hotspots.

The rest of the paper is organized as follows. Section 2 presents the related works. Section 3 identifies and formally defines our stochastic load balancing problem. Section 4 presents the overview and the detailed design of our load balancing scheme. Section 5 presents the performance evaluation of our scheme compared with other load balancing schemes in trace driven simulations. Finally, Section 6 concludes this paper and indicates our future work.

2 Related Works

Load balancing is a well-studied problem in distributed computer systems. Previous works [23], [31] consider the problem of statically balancing the load in order to minimize the mean job response time. For cloud, many works [4], [11], [12], [26], [29], [30], [32], [33], [36], [39] have been proposed to preform load balancing by VM migration from overloaded PMs to underloaded PMs.

Arzuaga *et al.* [4] present a load balancing VM migration framework based on a new metric for quantifying virtualized server load. The new metric is based on the variation in load measured on the PMs. In the framework, the load balancing algorithm chooses the VM migration that achieves the greatest improvement on this imbalance metric. Sallam *et al.* [26] consider the migration process as a multi-objective problem and propose a novel migration policy which utilizes a new elastic multi-objective optimization strategy to evaluate different objectives simultaneously. Tarighi *et al.* [32] propose a multi criteria decision method to migrate VMs between nodes. To reduce the time and cost to achieve load balance, Chen *et al.* propose RIAL [12], in which different weights are dynamically assigned to different resources based on their usage intensity in the PMs for determining which VM to migrate and where.

Singh *et al.* [30] propose a load balancing algorithm called VectorDot that takes into account the hierarchical and multi-dimensional resource constrains. It utilizes vectors to represent multi-dimensional resource demands. $ItemPathLoadFracVec(u)$ is the resource requirements of a virtual item on each node along u's flow path and $PathLoadFracVec(u)$ is the resource usage fraction of each node along the path. VectorDot migrates a VM u from an overloaded PM to the PM that has the lowest dot product of $ItemPathLoadFracVec(u)$ and $PathLoadFracVec(u)$. It attempts to avoid the node that is highly loaded on the resource for which the item requirements are also high. Wood *et al.* [36] propose a system named Sandpiper which automates the hotspot detection and VM migration. Sandpiper predicts the resource utilization on the PMs and compares it with a threshold to determine a hotspot. It uses the volume defined as the product of CPU, network and memory loads to capture the combined load on multidimensional

resources. Each VM has volume-to-size ratio (VSR) where size is the memory footprint of the VM, to measure the volume per unit byte moved. The VM migration algorithm attempts to migrate the VM with the maximum VSR.

Ye *et al.* [39] consider the live migration strategy of multiple virtual machines with different resource reservation methods. Shrivastava *et al*. [29] propose an application-aware virtual machine migration scheme, which takes into account the communication dependencies among VMs of a multi-tier enterprise application, the underlying data center network topology, as well as the capacity limits of the physical servers in datacenters. Chen *et al.* [11] propose a parallel migration to speed up the load balancing process, which migrates multiple VMs in parallel from overloaded hosts to under utilized hosts through solving the minimum weighted matching problem on a weighted bipartite graph.

Besides, many schemes [6], [8], [10], [16], [18], [20], [27], [28] balance load based on the prediction of future workloads of PMs or VMs. Chandra *et al.* [10] consider the problem of dynamic resource allocation for web applications running on the shared data centers. The resource allocation is based on a resource model whose parameters are continuously updated through an online monitoring and prediction framework. Gong *et al.* [18] use the signal processing techniques and statistical state-driven approaches to unobtrusively capture the patterns of the dynamic resource requirement and predict resource demands in the near future. Based on the previous work [18], Shen *et al.* [28] propose CloudScale, a system that elastically scales the resources of VMs according to their predicted demands and resolve scaling conflicts using VM migrations. Based on a Markov Chain model, Beloglazov [6] proposes an algorithm for host overload detection under the specified QoS goal which aims to maximize the mean intermigration time. Sharma *et al.* [27] present a cost-aware system for dynamically provisioning virtual server capacity, which combines the replication and migration mechanisms and the pricing model to select resource configuration and transition strategies to optimize the incurred cost. VirtualRank [16] observes multiple future load values to predict load tendency in the upcoming time slot and selects the potential migration target node using the Markov stochastic process.

Most aforementioned schemes make load balancing decisions based on deterministic estimations/predictions of resource usage. However, due to the dynamic workloads and the estimation/prediction errors, the resulted resource demand estimation may deviate significantly from the prediction, which can lead to SLA violations and requires further VM migrations. To address this issue, CloudScale [28] proposes adaptive padding and fast underestimation correction to handle the prediction error. But it still makes migration decisions based on deterministic estimation without considering the stochastic variances of resource usages of VMs and PMs. Our work [8] also aims to provide statistical guarantees of service quality taking into account the probability distribution of prediction errors. However, for multiple resources, it determines the destination PM for a VM migration by only checking whether the PM can accommodate the VM along each resource dimension separately. However, it may lead to improper migrations. Because a PM can be overloaded along one or more dimensions of resources, it is necessary to combine all resource dimensions together to consider the overall overload probability of a PM. Besides, it does not consider the hotspot detection in the context of statistical guarantees and the migration cost.

Compared with these works, in this paper we consider the stochastic characteristics of resource usages and propose a set of solutions for stochastic load balancing, including the prediction of probability distribution of resource demand, hotspot detection and VM migration algorithm considering the overall load status on multi-dimensional resources for PMs. In addition, our VM migration algorithm aims to minimize the migration cost that takes into account the network topology and improves the worst performance the system could experience from the hotspots.

3 Stochastic Load Balancing Problem

In this section, we study the workload uncertainty of VMs with a real world trace, formally define the problem of stochastic load balancing using VM migration, and discuss its hardness.

3.1 Workload Dynamics

In our stochastic load balancing problem, the resource demands and workloads are represented by random variables. Recent studies [7], [13], [22] show that VM's demands for certain resources are highly bursty and can be characterized by stochastic models. Previous works [14], [21], [35] assume the normal distribution for resource demands of VMs. In this paper we begin with a trace study to look into the VMs' workload dynamics and their probability distributions.

We use two traces: PlanetLab trace [2] and Google cluster trace [19]. The PlanetLab trace records the CPU utilization of VMs in PlanetLab platform every five minutes in 10 random days in March and April 2011. The Google Cluster trace records the CPU utilizations of tasks in about 11000 machines in a cluster in May 2011 for 29 days. We study the variance of CPU utilization and its empirical probability distribution during a short period in a day.

First, we examined the mean, 95th percentile, 5th percentile of CPU utilizations and its distribution for each VM in the PlanetLab trace. We chose 6 VMs that have typical workload distributions among all the examined VMs. Figure 1 and Figure 2 demonstrate the results of 6 VMs during 2.5 hours which are typical in our observations. Figure 1 shows the mean and percentile results of CPU utilizations. As we can see, although these VMs have different average CPU utilizations, their variances indicated by the error bars are quite large. For example, the CPU utilization of virtual machine 4 has mean 16% and varies from 7% to 25%. For each single virtual machine, we further looked into its distribution of CPU utilization within eight hours using the histogram of CPU utilization. We found that it approximately follows normal distribution, as exemplified by Figure 2.

We further examined the CPU utilization in the Google Cluster trace. Although the records in the trace are the resource usage of tasks instead of VMs, they reflect the workload dynamics in the real world applications and affect the virtual resource allocation when running in VMs. In fact,

we observed the similar results as in the PlantLab trace. Furthermore, the CPU utilizations of 6 typical VMs within 2.5 hours are shown in Figure 3. As we can see, the CPU utilization has a large variance. For VM 6, its mean CPU utilization is 20% and 95-th percentile is up to about 30%. The probability distributions of CPU is also exemplified by Figure 4, similar to normal distribution.

The above trace study indicates the highly dynamic resource demands of VMs in real world. Without considering the variance, only using deterministic estimation like mean value for VM migration could lead to resource overload and increase SLA violations. We analyze the adverse impact of such dynamic workloads on the performance of the load balancing schemes based on deterministic demand characterization in Section 4.4 and validate it in Section 5. Here we use the example shown in Figure 5 to illustrate the issue of the deterministic load balancing scheme for dynamic workloads. Suppose a cluster with four PMs and only CPU resource is considered for load balancing. PM 1 is determined as a hotspot and we choose to migrate VM 1 to other PMs to eliminate this hotspot. The CPU demand (in percentage) of VM 1 follows normal distribution $\mathcal{N}(40, 4)$. The CPU usage on PM 4 follows distribution $\mathcal{N}(40, 225)$ where the standard deviation is 15 which is quite common for highly dynamic workload according to our trace study. The CPU usages on PM 2 and 3 follow the distribution $\mathcal{N}(50, 25)$. If the load balancing scheme migrates VMs from the hotspot to the least loaded PM decided by the average CPU demand estimation as [36], VM 1 is migrated to PM 4 since it has the smallest mean CPU usage among the under-utilized PMs. However, if the VM is placed on PM 4, the probability of PM4 being overloaded is $\Pr(Agg > 100)$ where the aggregate CPU demand $Agg \sim \mathcal{N}(40 + 40, 4 + 225)$, which is equal to 10%. If VM 1 is placed on PM 2 or PM 3, the probability that they get overloaded is only 3% . We can see that the migration to PM 2 or PM 3 is a better choice for load balancing, which can reduce the SLA violations and possibly the number of the subsequent migrations. This simple example shows the impact of highly dynamic resource demands on the efficiency of load balancing.

Therefore, a load balancing scheme considering the stochastic resource demands is required to address this issue. According to our observation, in this paper we assume the resource demands follow the normal distribution for simplicity. Note that our stochastic load balancing scheme can be also used with other probability distributions.

3.2 Problem Description

Generally, a load balancing algorithm using VM migration needs to decide when to migrate VMs, which VM to migrate and where to migrate. In addition, the algorithm should minimize the total migration overhead. Since VMs can be migrated over multiple hops in the network, the migration overhead not only depends on the total amount of data transferred for VM migration, but also the number of hops the data being forwarded. As indicated in previous works [36], [38], the total migration overhead can significantly affect the performance of the applications inside the VMs, not only for migrated VMs but also for other VMs

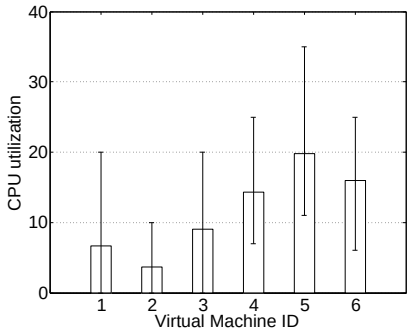


Figure 1. The variance of VM CPU utilization in PlanetLab trace.

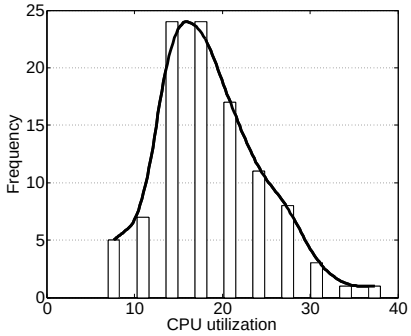


Figure 2. The probability distribution of VM CPU utilization in PlanetLab trace.

on the source PM and destination PM of the migration. By minimizing the data transmission cost of migration over the network, the load balancing algorithm can reduce the total migration time and thus the adverse impact on applications.

Therefore, our stochastic load balancing problem aims to make efficient VM migration decisions such that for each resource the total demand of VMs on each PM does not exceed the capacity of the resource of the PM with a high probability, while the migration overhead is minimized. We formally define this problem with the following analytical model. Table 1 describes the notations used in this paper.

We consider a data center consisting of M PMs and N VMs. We consider multiple types of resources including CPU, memory, disk I/O bandwidth and network bandwidth. For each resource r, its capacity on a PM i is denoted as c_i^r. The demand of a VM j for resource r is denoted as

Table 1
Notations

Notation	Description
M	The total number of PMs
N	The total number of VMs
c_i^r	The capacity of resource r on a PM i
D_j^r	The stochastic demand of VM j for resource r
s_j	The memory footprint of VM j
I_{ij}	The indicator variable. $I_{ij} = 1$ if VM j is placed on PM i
h_{ik}	The distance (hops) between PM i and PM k in the data center
ϵ	The threshold of the probability of a PM being overloaded

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.2016.2525984, IEEE Transactions on Cloud Computing

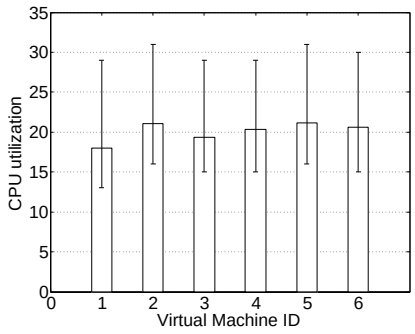


Figure 3. The variance of task CPU utilization in Google cluster trace.

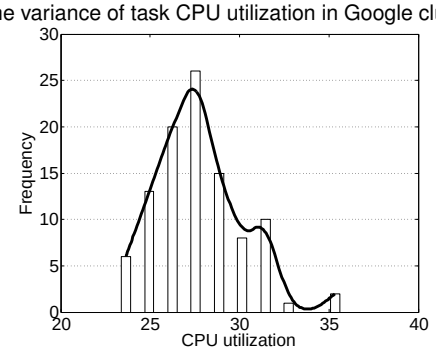


Figure 4. The probability distribution of task CPU utilization in Google cluster trace.

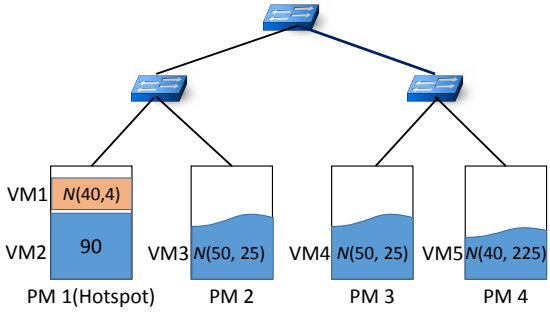


Figure 5. An example of load balancing with dynamic workloads.

a random variable D_j^r following a probability distribution that can be estimated from runtime measurement. Let I_{ij} be an indicator variable that is 1 if VM j is placed on PM i; otherwise $I_{ij} = 0$. Matrix $I = [I_{ij}]_{N\times M}$ is the VM placement matrix. The load balancing using VM migration is to compute an updated placement matrix $I^* = [I^*_{ij}]_{N\times M}$, with the constraint that on each PM i the total demand of VMs for each resource r does not exceed the capacity of that resource c_i^r with a high probability $1-\epsilon$, *i.e.*,

$$\Pr\left(\sum_j I^*_{ij} D_j^r \le c_i^r\right) \ge 1-\epsilon, \forall i, r \tag{1}$$

Here, ϵ is the threshold of the probability of a PM being overloaded. It indicates the risk of SLA violations on each PM and can be determined by SLA agreement.

The overhead of VM migration depends on the memory footprint of the VM to migrate and the distance between the source PM and the destination PM of the migration. Let s_j be the memory footprint (i.e., the number of bytes in MB or GB) of VM j, and h_{ik} ($h_{ik} \ge 1$) be the distance (*i.e.*, the number of hops) between PM i and PM k. The overhead of a VM migration is measured by the product of the memory footprint of the migrated VM and the distance between the migration source and destination, in $bytes \times hops$. Then, the total overhead of VM migrations needed to achieve the updated placement I^* can be computed by

$$\sum_j \sum_i \sum_k I_{ij} I^*_{kj} h_{ik} s_j \tag{2}$$

where $I_{ij}I^*_{kj} = 1$ if VM j is migrated from PM i to PM k, otherwise $I_{ij}I^*_{kj} = 0$.

Thus, the analytical formulation of our stochastic load balancing problem is

$$\text{minimize} \quad \sum_j \sum_i \sum_k I_{ij} I^*_{kj} h_{ik} s_j \tag{3}$$

$$\text{subject to} \quad \Pr\left(\sum_j I^*_{ij} D_j^r \le c_i^r\right) \ge 1-\epsilon, \forall i, r \tag{4}$$
$$I^*_{ij} = 0 \text{ or } 1, \forall i, j$$

This problem definition is not limited to any specific probability distribution for D_j^r, in order to provide a general stochastic framework to address the load balancing problem. To solve the problem, specific probability distribution functions need to be defined for the distribution estimation of resource demands and the calculation of Formula (1).

Example: Here we show an example of the above problem based on the Figure 5 with 5 VMs and 4 PMs. According to our above definition, we have

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The distance h_{ik} between PMs can be represented by

$$H = [h_{ik}]_{4\times 4} = \begin{bmatrix} 0 & 2 & 4 & 4 \\ 2 & 0 & 4 & 4 \\ 4 & 4 & 0 & 2 \\ 4 & 4 & 2 & 0 \end{bmatrix}$$

Suppose $\epsilon = 0.05$. Then, based on the calculation in Section 3.1, PM 2 and PM 3 are appropriate candidates of migration destination for VM 1 since the probability that they get overloaded after VM 1 is migrated to either of them is 3%, less than ϵ. To minimize the migration cost that depends on the distance between the source PM and the destination PM, the best choice is to migrate VM 1 to PM 2 because PM 2 is closer to PM 1. Thus, the optimal solution, described by the new placement matrix, is

$$I^* = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Generally, this problem is NP-hard, since it can be regarded as a variant of the stochastic knapsack problem [17]. In addition, due to the large numbers of VMs and PMs in

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.2016.2525984, IEEE Transactions on Cloud Computing

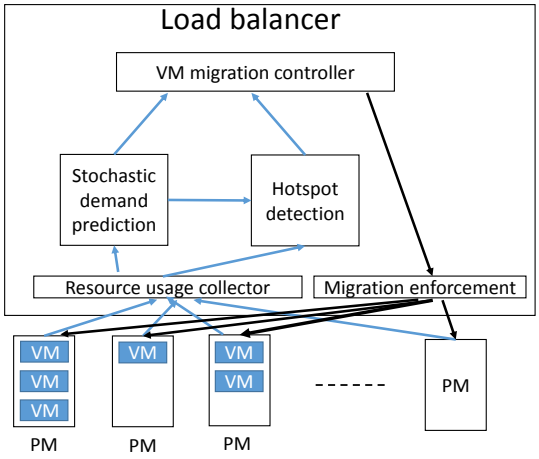


Figure 6. The overview of the stochastic load balancing scheme.

a data center, the problem size is also significant. Considering such computational complexity, in practice, the cluster load balancers deployed in the data center mostly rely on heuristics to solve the problem. Thus, we develop a heuristic algorithm to solve our stochastic load balancing problem.

4 Load Balancing Scheme

In this section, we describe our load balancing scheme. Our scheme has common components as other previous load balancing schemes [4], [6], [8], [10], [12], [18], [28], [30], [36], including the profiling of resource demand, hotspot detection and hotspot migration. But as opposed to these previous works, in our load balancing scheme, the goal of the profiling is to generate the stochastic characterization, *i.e.*, the probability distribution of the resource demand and usage. Previous load balancing schemes give deterministic estimations which cannot capture the uncertainty of workloads. For hotspot detection, we propose to examine the probabilistic guarantee in Formula (1) based on the probability distribution of resource demands of VMs. For hotspot migration, a heuristic algorithm is proposed to solve the problem formulated in Section 3.2 to decide efficient VM migrations.

Figure 6 shows an overview of our load balancing scheme and the information flow among different components. The scheme is centralized. A load balancer runs in a central server. In each PM the monitor tracks the CPU, memory and network usage of each VM and PM, and periodically sends the usage statistics to the resource usage collector of the load balancer. The resource usage information is used by the stochastic demand prediction component to estimate the distribution of resource demand for each VM. The distribution estimations are then used for the hotspot detection and the VM migration. The migration enforcement receives the instructions from the VM migration controller to do the migrations.

The load balancing scheme proposed here assumes the normal distributions for resource demands, according to the results of our previous trace study in Section 3.1. The distribution estimation of stochastic resource demands and the computation of overloading probability of PMs are based on the normal distribution. Nevertheless, our scheme can be easily extended to other types of distributions of resource demands, with specific distribution estimation and probability computation for the given type of distribution functions.

4.1 The Profiling of Stochastic Resource Demand

For the profiling, each VM's resource usages on CPU, memory, network bandwidth and disk I/O are monitored. The load balancer receives periodic reports of resource usages from each PM, and repeatedly estimates the probability distributions of resource demands of VMs in a sliding window.

According to our trace study in Section 3.1, we assume that the demands of the VMs for each resource r follow normal distribution. For VM j, supposing that its demand for resource r follows $\mathcal{N}(\mu_j^r, {\sigma_j^r}^2)$, the problem is how to estimate parameters μ_j^r and ${\sigma_j^r}^2$ based on the previous usage observations. A straightforward way is to compute the sample mean and the sample variance of the observations in the window and use them as the estimations of the parameters of the normal distribution. However, using the statistics of the historical observations are not good predictions for the resource demand in the future because they cannot capture the increasing or decreasing trends of resource utilizations. Therefore, we propose a method which integrates the time series prediction techniques into the estimations of the distribution parameters of resource demands, while not relying on any specific time-series prediction technique.

Suppose that a time-series prediction function makes a prediction to the demand in the next time interval based on n prior resource utilization observations $o_1^r, o_2^r, \ldots o_n^r$ for resource r, as in [8], [36], [37]. The function can be any of Auto Regressive (AR) models, exponentially weighted moving average or any other time-series prediction technique. The function output is the prediction of $\hat{o}_{n+1}^r$, which is a constant.

In addition, we evaluate the estimation errors for most recent n observations, that is, $\varepsilon_1^r = o_1^r - \hat{o}_1^r$, $\varepsilon_2^r = o_2^r - \hat{o}_2^r$, $\ldots$, $\varepsilon_n^r = o_n^r - \hat{o}_n^r$. We calculate the sample mean μ_ε and the sample variance σ_ε^2 of these estimation errors $\varepsilon_1^r, \ldots, \varepsilon_n^r$ as follows:

$$\mu_\varepsilon = \frac{1}{n}\sum_{i=1}^{n}\varepsilon_i \tag{5}$$

$$\sigma_\varepsilon^2 = \frac{1}{n}\sum_{i=1}^{n}(\varepsilon_i - \mu_\varepsilon)^2 \tag{6}$$

Then, we estimate the statistics μ_j^r and ${\sigma_j^r}^2$ of the resource demand in the time interval $n+1$ as

$$\mu_j^r = \hat{o}_{n+1}^r + \mu_\varepsilon \tag{7}$$

$${\sigma_j^r}^2 = \sigma_\varepsilon^2 \tag{8}$$

Taking into account the constant prediction $\hat{o}_{n+1}^r$, the above estimation to the distribution of the resource demand captures both the changing trend of the resource demand and the uncertainty (variance). Besides, considering that the distribution of VMs' workload can change with the time frequently, continuous estimation of workload distribution for the next time interval can capture such changes timely and help to effectively detect and avoid hotspots.

4.2 Hotspot Detection

To detect hopspots, our scheme periodically evaluates the resource allocation status of each PM based on the predicted distribution of resource demands of VMs on the PM. Previous works with deterministic demand prediction determine a hotspot PM by checking whether the aggregate resource demand of VMs on the PM exceeds a threshold. Instead, our stochastic load balancing scheme determines a hotspot by checking whether the probability of overloading is no larger than ϵ for each resource r, *i.e.*, whether Formula (1) holds.

Let V be the set of VMs running on a PM i. For each VM j, the demand for resource r D_j^r follows the distribution $\mathcal{N}(\mu_j^r, \sigma_j^{r2})$ which is predicted by the profiling in Section 4.1. Then, because we assume that each VM's demand is independent of others, the aggregate demand $\sum_{j\in V} D_j^r$ follows the distribution $\mathcal{N}(\sum_{j\in V}\mu_j^r, \sum_{j\in V}\sigma_j^{r2})$ according to the property of normal distribution. Then, it is easy to show that $\Pr(\sum_{j\in V} D_j^r \le c_i^r) \ge 1-\epsilon$ is equal to

$$\frac{c_i^r - \sum_{j\in V}\mu_j^r}{\sqrt{\sum_{j\in V}\sigma_j^{r2}}} \ge \Phi^{-1}(1-\epsilon) \tag{9}$$

where $\Phi(*)$ is the inverse of the cumulative distribution function of the standard normal distribution. PM i is determined as a hotspot, if there is any resource r for which Inequality (9) does not hold. Alternatively, to avoid unnecessary migrations caused by small transient spikes of workload, similar to previous work [36], we can determine a hotspot only if the above probabilistic guarantee for any resource is violated for a sustained time. That is, a PM is determined as a hotspot if Inequality (9) does not hold in at least k out of the n most recent checks with the predicted demands as well as in the current check.

4.3 Hotspot Migration

After hotspots are identified, the load balancer needs to solve the problem of which VMs to migrate and where in order to dissipate the hotspots. This problem is formulated in Section 3.2 and its hardness is shown. In this section, a heuristic hotspot migration algorithm is proposed.

For each hotspot PM i, based on the estimated resource demands of VM set V on i, we can obtain $\sum_{j\in V} D_j^r \sim \mathcal{N}(\sum_{j\in V}\mu_j^r, \sum_{j\in V}\sigma_j^{r2})$ and then compute the probability $\Pr(\sum_{j\in V} D_j^r > c_i^r)$ for each resource r, denoted by $P_{i,V}^r$. Because i is a hotspot, there must exist one or more resources for which $P_{i,V}^r > \epsilon$. Such resources are called *tense resources*. We define the *overload risk* of a hotspot i as the probability of at least one of tense resources being overloaded.

$$overloadrisk(i) = 1 - \prod_{r\in R}(1 - P_{i,V}^r) \tag{10}$$

where R is the set of tense resources. As we can see, the higher overload risk indicates the higher probability of PM i being overloaded on any of tense resource. Note that, Formula (10) assumes that the utilizations of different resources are independent. This could be not true if, for example, an application can be both CPU intensive and I/O intensive and in this case the overloading probabilities of these two resources are correlated. Because the correlations between multiple resources vary with the applications and are hard to characterize, we assume the independency among different resources and simply regard $overloadrisk$ as a metric to measure the overall resource tensity in a PM.

4.3.1 Algorithm Overview

In our hotspot migration algorithm, the hotspots are sorted in decreasing order of their overload risks and stored in a list. The algorithm starts from the PM with the largest overload risk among the hotspots, selects the VM whose removal can most efficiently reduce the PM's overload risk, and determines which PM this VM is migrated to. If no VMs can be migrated from the hotspot PM with the largest overload risk, the algorithm examines the next hotspot in the list. The algorithm hypothetically conducts VM migration, updates the resource allocations of source PM and destination PM, and recomputes the overload risk of the remaining hotspots. The algorithm iteratively runs over the remaining hotspots, and terminates until there are no hotspots left. The determination of the destination PM for a VM migration needs to ensure the constraint (4) while minimizing the total migration cost given in (3). For a VM to be migrated, we perform a hypothetical migration to every under-utilized PM, and select a PM as a destination candidate if at this PM the resources still satisfy constraint (4) after the migration. The PM with the minimum migration cost among all the candidates is chosen as the final destination PM in order to minimize the total migration cost (3).

The algorithm records all the hypothetical VM migrations (which VM is selected and where to migrate) determined in the way described above and the final output is a list of VMs and their destination PMs. Note that during the iteration, the algorithm may not find any feasible VM migration to reduce the load for any hotspot when all the PMs become heavily loaded. In this case, to avoid infinite loop, the algorithm also ends and outputs the VM migrations obtained before termination. Next, we explain how the algorithm determines which VM to migrate and where.

4.3.2 Which VM to migrate and where

To decide which VM to migrate, we introduce a new metric for a VM which indicates the degree of overload risk reduction caused by its removal, denoted by *ORreduction*. Suppose that a hotspot PM i has the VM set V on it. Given the estimations of VMs' demand distributions for resource r, we can compute the aggregate demand distribution after the removal of VM $j \in V$, and thus the following probability

$$P_{i,V\setminus\{j\}}^r = \Pr\left(\sum_{k\in V\setminus\{j\}} D_k^r > c_i^r\right) \tag{11}$$

We define

$$S_{i,V\setminus\{j\}}^r = \begin{cases} P_{i,V\setminus\{j\}}^r & \text{if} \quad P_{i,V\setminus\{j\}}^r > \epsilon \\ \epsilon & \text{if} \quad P_{i,V\setminus\{j\}}^r \le \epsilon \end{cases} \tag{12}$$

Then, the ORreduction for the removal of VM j from PM i is computed as

$$ORreduction_i(j) = overloadrisk(i) - \left(1 - \prod_{r \in R}(1 - S^r_{i,V\setminus\{j\}})\right) \quad (13)$$

Let $S^r_{i,V\setminus\{j\}} = \epsilon$ when $P^r_{i,V\setminus\{j\}} \leq \epsilon$. The reason for that is, for two VMs j and k, if either one of their removals can ensure resource r satisfies the probabilistic guarantee for overloading in Formula (1), they contribute the same to the overload risk reduction along the dimension of resource r. In this way, if either one of their removals can have all tense resources to achieve the probabilistic guarantee, VM j and k have the same overload risk reduction, *i.e.*, $ORreduction_i(j) = ORreduction_i(k)$. From the definition we can see $ORreduction$ captures the load along each resource dimension and measures the benefit of a VM migration for alleviating the tensity of resources.

The VMs on the currently considered hotspot PM are sorted in a list in the decreasing order of $ORreduction$. Furthermore, the VMs in the list are divided into different groups by different intervals of $ORreduction$. Given Formula (12) and Formula (13), the maximum value of $ORreduction$ in the list is $Max_{ORreduction} = overloadrisk(i) - (1 - \prod_{r \in R}(1 - \epsilon))$. Then, the VMs having the maximum value of $ORreduction$ form the first group, denoted by V_{G1}. Starting from $Max_{ORreduction}$, the range of $ORreduction$ is divided into intervals of equal width α ($\alpha > 0$), and the VMs with their $ORreduction$ in the same interval form a group. For example, the VMs in the second group V_{G2} have their $ORreduction$ falling to the interval $(Max_{ORreduction}, Max_{ORreduction} - \alpha]$. The last group V_{Gn} falls into the interval $(Max_{ORreduction} - (n-2)\alpha, Max_{ORreduction} - (n-1)\alpha]$ where $Max_{ORreduction} - (n-2)\alpha > 0$. By dividing the VM list in terms of different $ORreduction$ intervals, the VMs with the similar gain for overload reduction are classified into a same group. Note that the removal of any VM in V_{G1} would let the PM achieve probabilistic guarantee (1) for each resource r. Thus, V_{G1} is considered at first.

In this way, to choose a VM to migrate, the VM groups are examined in the descending order of $ORreduction$ intervals and in each group the VM which can achieve the smallest migrate overhead is preferred for migration. In a same group, for every VM j, we check all other PMs and determine a destination PM M_j that can accommodate VM j's resource demand while having the smallest migration overhead for the VM among all the PMs, denoted by $Cost(j, M_j)$. Given a PM l and the VM set V_l on it, the PM l can accommodate a VM j with resource demand D^r_j from another PM i if and only if $\Pr(D^r_j + \sum_{k \in V_l} D^r_k \leq c^r_l) \geq 1 - \epsilon$. The migration overhead is the product of the VM's memory footprint s_j and the distance h_{li} between the destination PM l and the source PM i, that is, $s_j h_{li}$. Finally, the VM j with the smallest $Cost(j, M_j)$ among all the VMs in the group is selected to migrate and M_j is its destination PM.

Here, α controls the degree of $ORreduction$ similarity in a group. In the extreme case that α is small enough and each single VM forms a group, the VM selection actually is the largest $ORreduction$ first strategy regardless of the migration cost. Oppositely, if α is large enough and all the VMs are in one group, the VM selection becomes the smallest migration cost first strategy. Therefore, α indicates the trade-off between the optimality of the overload risk reduction and the optimality of the migration cost. Appropriate value of α can be determined through experiments. Algorithm 1 shows the pseudo code of the hotspot migration algorithm.

As we can see, each time our algorithm determines one VM migration for the most overloaded hotspot, and then resorts all the hotspots in the order of their overload risks "after this migration" to find the next possible migration on the possibly new most overloaded hotspot. Previous works either continuously find VM migrations for the most overloaded hotspot until it is not overloaded or determine one VM migration for each hotspot in order of their overload degree but without resorting the hotspots after determining a migration. In contrast, our algorithm always tries to improve the worst hotspot after each migration and thus can fairly reduce the loads on all the hotspots to the same level of load status. With such fairness, even when not all the hotspots can be eliminated, our algorithm is able to improve the worst performance the system could experience from the hotspots.

4.3.3 Time complexity analysis

We assume that the number of resource types is bounded by a constant, which is true in practice. Let v_{max} be the maximum number of VMs a PM can run. According to our algorithm, each iteration on the while loop of line 2 has to find a new VM migration otherwise the loop ends. This indicates the number of iterations on the while loop of line 2 is at most $v_{max}|H|$ where $|H|$ is the number of hotspots initially detected. In each iteration, line $4 \sim 10$ have time complexity $O(|H| \log |H|)$. Line $11 \sim 39$ iterate over $|H|$ hotspots. During each iteration in them, line $12 \sim 14$ have time complexity $O(v_{max} \log v_{max})$ for each VM the algorithm searches the PM that can accommodate it and has the minimum migration cost, which totally incur $v_{max}|U|$ running time where $|U|$ is the total number of PMs. Thus, line $11 \sim 39$ have total time complexity $O(|H|v_{max}(\log v_{max} + |U|))$. Then, the time complexity of the total algorithm is $O(v_{max}|H|^2(\log |H| + v_{max}(\log v_{max} + |U|)))$. Considering that $\log |H|$ and $\log v_{max}$ is much smaller than other items, the time complexity can be simplified as $O(v^2_{max}|H|^2|U|)$. Because v_{max} is usually a small limited number and the set of the hotspots detected simultaneously is only a small part of the PMs in the datacenter, the running time is proportional to the size of the datacenter. This indicates the computation efficiency of our heuristic algorithm, which is important for a load balancer to make timely decisions.

4.4 Performance analysis and comparison

Most previous works [6], [10], [18], [28], [37] decide VM migrations for load balancing based on deterministic prediction of resource demands of VMs. To simplify our analysis, we assume a deterministic load balancing scheme making decisions based on the deterministic estimation of demand for resource r by μ^r_j in Formula (7). This load balancing scheme tries to migrate VMs on the hotspot until it is not overloaded.

Algorithm 1 Hotspot Migration Algorithm.

Input: U-the set of all PMs, H-the set of hotspot PMs, V_i-the set of VMs on every PM i, the demand distributions of all VMs,

Output: S_m-the set of mappings between the VMs to be migrated and the destination PMs

$S_m \leftarrow \emptyset$;
while $H \neq \emptyset$ **do**
  bool $newVMmigration \leftarrow false$;
  for all PM $i \in H$ **do**
    $R_i \leftarrow \emptyset$;{R_i is the set of tense resources on PM i}
    for all resource r **do**
      if $\Pr(\sum_{j \in V} D_j^r > c_i^r) > \epsilon$ **then**
        Add r to R_i
    Compute $overloadrisk(i)$;
  Sort H in decreasing order of $overloadrisk$;
  for PM i=1,2,...in H **do**
    for all $j \in V_i$ **do**
      Compute $ORreduction_i(j)$ based on R_i;
    Sort V_i in decreasing order of $ORreduction$
    Divide V_i into groups V_{G1}, V_{G2}, ..., V_{Gn} by $ORreduction$ intervals of width α;
    for $X = V_{G1}, V_{G2}, \ldots, V_{Gn}$ **do**
      $j_{min} \leftarrow \emptyset$; $mincost \leftarrow +\infty$;
      for all $j \in X$ **do**
        $C_j \leftarrow \emptyset$; {C_j stores the candidate PMs that can accommodate VM j}
        for all $l \in U \setminus H$ **do**
          if $\forall r \Pr(D_j^r + \sum_{k \in V_l} D_k^r \leq c_l^r) \geq 1 - \epsilon$ **then**
            $C_j \leftarrow C_j \cup \{l\}$;
        if $C_j \neq \emptyset$ **then**
          for all $l \in C_j$ **do**
            Compute $Cost(j, l) = s_j h_{il}$;
          $M_j \leftarrow \arg\min_l Cost(j, l)$; { find the PM that has minimum migration cost for VM j}
        if $mincost > Cost(j, M_j)$ **then**
          $j_{min} \leftarrow j$; $mincost \leftarrow Cost(j, M_j)$;
      if $j_{min} \neq \emptyset$ **then**
        Add migration mapping $(j_{min}, M_{j_{min}})$ to S_m;
        $newVMmigration \leftarrow true$;
        Update the resource allocation status on PM i and the destination PM with the hypothetical migration;
        if PM i achieves the probabilistic guarantee on every resource **then**
          $H \leftarrow H \setminus i$;
          Break;
    if $newVMmigration = true$ **then**
      break;
    else
      Return S_m;
Return S_m;

Suppose a PM i has x VMs with demand $D_j^r \sim \mathcal{N}(\mu_j^r, \sigma_j^{r\,2})$ $(1 \leq j \leq x)$ for resource $r \in R$ (R is the set of resource types in a PM). The goal of a deterministic scheme is to ensure $\sum_{i=1}^{x} \mu_j^r < c_i^r$. In the case of $\sum_{i=1}^{x} \mu_j^r < c_i^r$, we are interested in the conditional probability $\Pr(\sum_j D_j^r < c_i^r | \sum_{i=1}^{x} \mu_j^r < c_i^r)$, which indicates the efficiency of the deterministic load balancing scheme. Because $\sum_j D_j^r \sim \mathcal{N}(\sum_j \mu_j^r, \sum_j \sigma_j^{r\,2})$, we can easily know $\Pr(\sum_j D_j^r < c_i^r | \sum_{i=1}^{x} \mu_j^r < c_i^r) > 0.5$. It also indicates that with the deterministic scheme, the upper bound of the probability of resource r being overloaded is 0.5, which is rather high. According to Formula (10), the overload probability for the PM is $1 - \prod_{r \in R} (1 - P_{i,V}^r)$ where $P_{i,V}^r$ is the probability of resource r being overloaded. Suppose $|R| = 4$, then the upper bound of overload risk for the PM is as high as 0.93. It indicates that the PM can be still overloaded with a high probability even if the load balancer achieves its goal based on its deterministic estimation. It also reflects the adverse impact of highly dynamic resource demand on the efficiency of the load balancing.

In contrast, our stochastic load balancing scheme can limit the upper bound of overload risk by the probabilistic guarantee against the overloading for each resource. Given ϵ, according to Formula (1), the upper bound of overload probability for the PM is $1 - (1 - \epsilon)^{|R|}$. When $\epsilon = 0.05$, the upper bound is 0.185, which is much smaller than 0.93.

The value of parameter ϵ controls the overload probability, and also affects the resource utilization. Smaller ϵ requires a PM to reserve more resources to accommodate the possible variances of resource demands, which may lead to less resource multiplexing efficiency. It can be easily illustrated by the following inequality which is equivalent to Formula (1).

$$c_i^r \geq \sum_{j \in V} \mu_j^r + \Phi^{-1}(1 - \epsilon) \sqrt{\sum_{j \in V} \sigma_j^{r\,2}} \tag{14}$$

where $\Phi^{-1}(1 - \epsilon)$ increases with ϵ. To ensure this inequality with a smaller ϵ, some VMs may have to be migrated out from the PM. Then, fewer VMs share the PM and thus the resource utilization decreases. The previous deterministic schemes do not involve the variance (the second term) on the right side of the above inequality.

5 Performance Evaluation

We conducted trace-driven experiments on CloudSim [9] to evaluate the performance of our proposed stochastic load balancing algorithm in a three-resource environment (*i.e.*, CPU, memory and bandwidth). We used the VM utilization trace from PlanetLab [2] and Google Cluster [19] to generate VM workload. We implemented our stochastic load balancing algorithm in the simulator, represented by SLB. We study the performance in terms of the number of VM migrations, the number of overloaded PMs, the PM resource consumptions, the speed of load balancing and the total performance degradations. We use Sandpiper [36] to represent the reactive load balancing algorithms and use CloudScale [28] to represent the proactive load balancing algorithms.

We simulated the cloud datacenter with 1000 PMs hosting 5000 VMs. The PMs are modeled from commercial product HP ProLiant ML110 G4 servers (1860 MIPS CPU, 4GB memory) and the VMs are modeled from EC2 micro instance (0.5 EC2 compute unit, 0.633 GB memory, which is equivalent to 500 MIPS CPU and 613 MB memory). The CPU utilization trace from PlanetLab VMs and Google Cluster VMs, and memory utilization trace

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.2016.2525984, IEEE Transactions on Cloud Computing

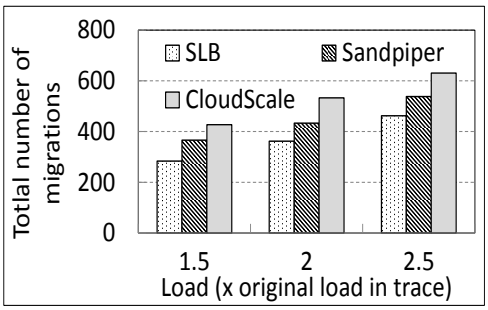


Figure 7. The number of VM migrations using PlanetLab trace.

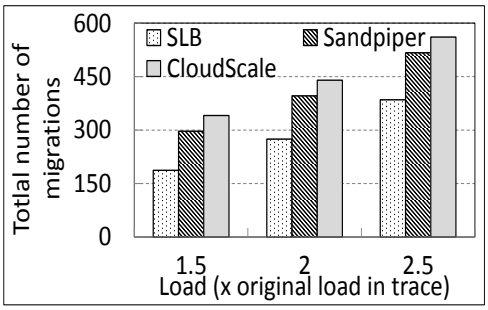


Figure 8. The number of VM migrations using Google Cluster trace.

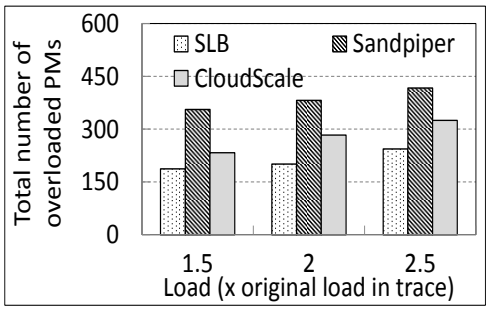


Figure 9. The number of overloaded PMs using PlanetLab trace.

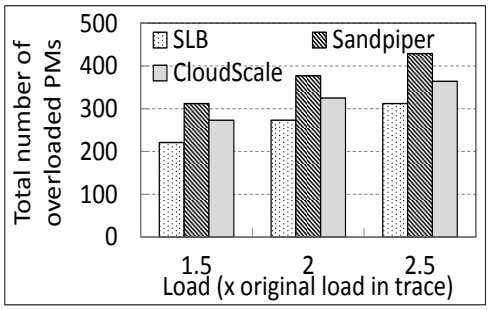


Figure 10. The number of overloaded PMs using Google Cluster trace.

from Google Cluster VMs are used to drive the VM CPU and memory utilizations in the simulation. To simulate bandwidth usage, as in [29], we generated 5 different groups of (mean, variance range) for bandwidth 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 bandwidth utilization to a value generated by a randomly chosen group. We increased the VM's workload to 1.5, 2 and 2.5 times of its original workload in the trace to study the performance under various workload levels. At the beginning, the VMs are randomly allocated to the PMs. We used this VM-PM mapping for different load blanching algorithms in each experiment to have fair comparison. When the simulation is started, the simulator updates the resource utilization status of all the PMs in the datacenter every 5 minutes according to the traces, and records the number of VM migrations, the number of overloaded PMs (the occurrence of overloaded PMs) and the resource utilization of all PMs and VMs during that period. We used 0.75 as the resource utilization threshold for CPU memory and bandwidth usage to determine whether the PM is overloaded. Sandpiper and CloudScale perform VM migrations whenever a PM is detected overloaded (*i.e.*, either CPU, memory or bandwidth utilization exceeds 0.75) and select the destination PM based on their corresponding PM selection algorithms. In SLB, an overloaded PM chooses the VM based on the stochastic model, and the migrating VM chooses the destination PM based on the stochastic model and network topology. The profiling of stochastic resource demand in SLB uses $n = 30$ most recent measurements.

5.1 The number of VM migrations

Figure 7 and Figure 8 show the total number of VM migrations with varying workload ratios. Figure 7 and Figure 8 show the experimental results with the PlanetLab trace and Google Cluster trace, respectively. For each workload ratio, the number of VM migrations follows SLB<Sandpiper<CloudScale. SLB outperforms Sandpiper and CloudScale because each PM can find the VM that is expected to effectively release PM workload with the highest probability, and also can find the most suitable destination PM to host the migrating VM, resulting in a reduced need of VM migrations in a long run. That is, SLB is able to keep a long-term load balance state while triggering a smaller number of VM migrations than the alternative methods. Also, SLB 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 CloudScale in the same period of time. CloudScale generates a larger number of VM migrations than Sandpiper in each round because CloudScale migrates VMs not only for a correctly predicted overloaded PM but also for an incorrectly predicted overloaded PM, but Sandpiper only migrates VMs for occurred overloaded PMs. The results under the PlanetLab trace have higher numbers of VM migrations than the results under the Google Cluster trace for two reasons. First, the PlanetLab trace has a relatively higher workload level than the Google Cluster trace, meaning that the average utilization of the PMs is more close to 0.75. Second, the workload in PlanetLab is more fluctuant and tends to lead to inaccurate predictions of the load balancing algorithms.

5.2 The number of overloaded PMs

In this experiment, each PM checks its load status every 5 minutes during the simulation. A PM is regarded as overloaded if the utilization of either its CPU, memory or bandwidth resource exceeds the predefined threshold.

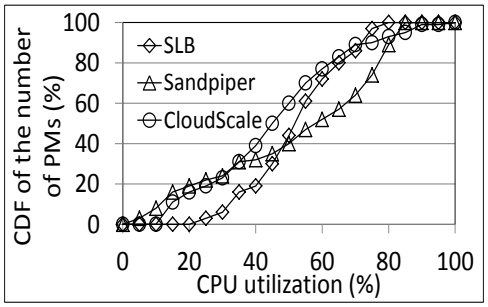


Figure 11. CDF of PM CPU utilization after load balancing.

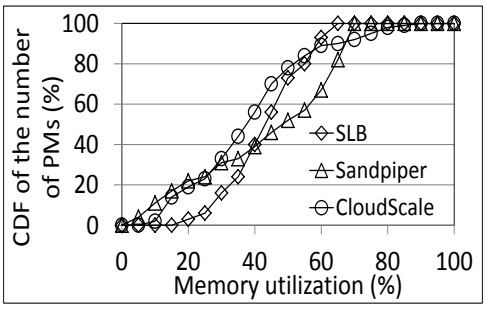


Figure 12. CDF of PM memory utilization after load balancing.

Figure 9 and Figure 10 show the cumulative number of overloaded PMs detected in the system during the 24 hour simulation, with respect to the PlanetLab trace and Google Cluster trace. The number of overloaded PMs increases with workload ratio, and follows SLB<CloudScale<Sandpiper for each workload ratio. SLB outperforms the other two with fewer overloaded PMs since it uses the stochastic model for both the migrating VM selection and the destination PM selection, to maintain a long-term load balance state. CloudScale produces fewer overloaded PMs than Sandpiper because its predicted overloaded PMs migrate VMs out before they become overloaded, while Sandpiper conducts VM migrations upon the PM overload occurrence. The numbers for the PlanetLab trace are higher than those for the Google Cluster trace due to the same reasons mentioned in Section 5.1.

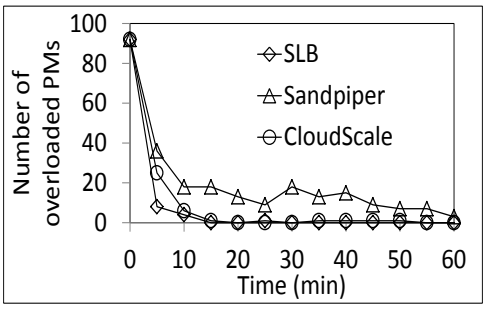


Figure 13. The number of overload PMs under workload of ratio 1.5.

5.3 PM resource utilizations

Figure 11 and Figure 12 present the cumulative distribution function (CDF) of the number of PMs versus the CPU and memory utilizations, after the first load balancing with the PlanetLab trace in the three methods. The figures show that SLB results in a similar resource utilization distribution across the PMs and they keep the CPU and memory utilizations of all the PMs under the threshold. Due to a long-term load balance state maintenance caused by stochastic model based hotspot detection and hotspot migration, SLB is able to keep all the PMs in a medium resource utilization of around 50%. The CDF show that SLB achieves a more balanced status for the system than the other two methods. Due to the fluctuation of VM loads and inappropriate VM migrations, Sandpiper and CloudScale sometimes fail to achieve this goal. For Sandpiper, around 20% of the PMs exceed CPU utilization threshold and around 10% of the PMs exceed memory utilization threshold. For CloudScale, around 10% of the PMs exceed CPU utilization threshold and around 10% of the PMs exceed memory utilization threshold.

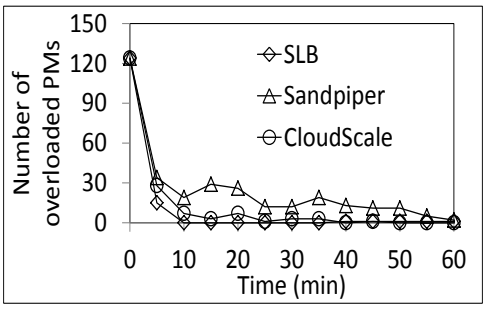


Figure 14. The number of overload PMs under workload of ratio 2.

5.4 The speed of load balancing

We then measure the temporal distributions of the cumulative number of overloaded PMs in the system to measure the capability of the load balancing methods in preventing overloading PMs in the long-term. Figure 13, Figure 14 and Figure 15 show the CDF of the number of overloads over time with the workload ratio of 1.5, 2 and 2.5, respectively. Since

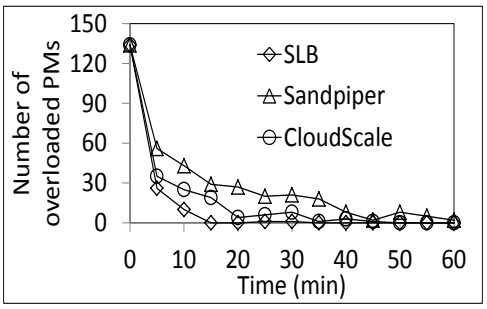


Figure 15. The number of overload PMs under workload of ratio 2.5.

we used the same random initial placement, the numbers of overloaded PMs in the beginning are exactly the same for the three balancing methods. When the workload is low, as Figure 13 shows, the number of overloaded PMs generated initially accounts for 20% of the total number of overloaded PMs in Sandpiper. The CDF curve of Sandpiper indicates that PM overload can happen at any moment during the simulation time. This is because Sandpiper only focuses on eliminating current overloads, which tends to generate future overloaded PMs due to the dynamically varying workload requests from the VMs. CloudScale and SLB not only eliminate the overloaded PMs generated initially but also prevent overloading PMs. We see that the number of overloaded PMs generated initially accounts for 80% of the total number of overloaded PMs in CloudScale, and only 20% are generated during the subsequent time in the 1 hour. SLB outperforms CloudScale due to its more accurate stochastic model with sophisticated mathematical tools. Figure 14 and Figure 15 show similar distribution as in Figure 13 due to the same reasons. The results confirm that SLB is effective in preventing overloading PMs with different workload levels.

5.5 The VM performance degradation

When a VM is being migrated to another PM, its performance (response time) is degraded [34]. We also aim to minimize the VM performance degradation caused by migrations. We calculate the performance degradation D of a VM that migrates to PM based on a method introduced in [5], [34]. $D = \sum d \cdot \int_t^{t+\frac{M}{B}} u(t)\mathrm{d}t$, where t is the time when migration starts, M is the amount of memory used by V, B is the available network bandwidth, $\frac{M}{B}$ indicates the time to complete the migration, $u(t)$ is the CPU utilization of V, and d is the migration distance from the source PM to the destination PM. The distance between PMs can be determined by the cloud architecture and the number of switches across the communication path [24], [25].

Figure 16 and Figure 17 show the total performance degradation $\sum D$ of the three methods for the PlanetLab and Google Cluster traces, under 1.5x, 2x and 2.5x of workload, respectively. We see that the total performance degradation of SLB is lower than that of CloudScale and Sandpiper in both traces. This is caused by the distinguishing features of SLB. First, SLB triggers fewer VM migrations. Second, SLB tries to minimize performance degradation in destination PM selection by considering network topology. Third, SLB chooses VMs with lower utilizations. Sandpiper generates lower performance degradation than CloudScale because it generates fewer VM migrations as shown in Figure 7 and Figure 8. We also see that in both traces, the performance degradation of the three methods follows SLB<Sandpiper<CloudScale, and the difference is small in the Google Cluster trace.

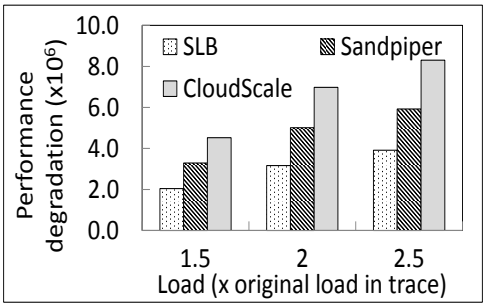


Figure 16. Total VM performance degradation using PlanetLab trace.

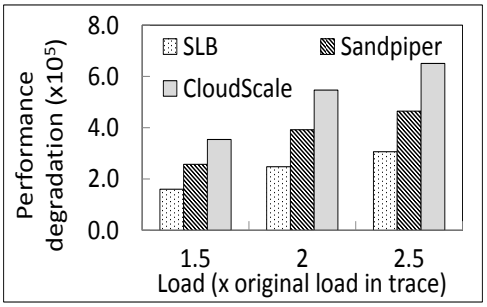


Figure 17. Total VM performance degradation using Google Cluster trace.

6 Conclusion

In this paper, we consider the VM migration based load balancing problem with highly dynamic resource demands. By our trace study, we demonstrate that in real world the resource utilization of VMs are highly dynamic and bursty. The previous load balancing schemes detect hotspots and decide VM migrations based on deterministic resource demand prediction, which can lead to poor performance and severe SLA violations. To address this issue, we propose a stochastic load balancing scheme. With characterizing the resource demands of VMs as random variables, our scheme provides the probabilistic guarantee against resource overloading, that is, the aggregate VM demand for any resource in a PM does not exceed its capacity with a high probability. To this end, it addresses the prediction of the probability distribution of resource demands, and determines hotspots and VM migrations with stochastic characterizations of resource demands. The VM migration algorithm in the scheme aims to minimize the migration cost for load balancing considering the network topology and improves the worst performance the system could experience from the hotspots. Our trace-driven experiments demonstrate the efficiency and the advantages of our scheme compared with the previous deterministic load balancing schemes.

In the future, we will investigate the distribution of VM demands in a large-scale, and evaluate the impact of different distributions of workloads on the performance of load balancing. Specifically, we will look into the exponential distribution and its impact on the efficiency of deterministic load balancing schemes. We will extend the stochastic load balancing scheme considering different probability distributions. In addition, although our VM migration algorithm considers the network topology by taking into account the distance from the source to the destination PM for a VM migration, it does not consider the bandwidth usages on the links of the migration path. In practice, the link congestion on a path can prolong migration time. On the other hand, many applications run on multiple VMs in a distributed manner. To ensure application performance, the bandwidth demand among these VMs have to be satisfied.

Thus, determining the destination of VM migrations need to consider the available bandwidth on the paths between the destinations and other VMs used by the application.

REFERENCES

[1] Microsoft Azure. http://www.windowsazure.com.

[2] planetlab workload traces. https://github.com/beloglazov/planetlab-workload-traces.

[3] Amazon Web Service. http://aws.amazon.com/.

[4] E. Arzuaga and D. R. Kaeli. Quantifying load imbalance on virtualized enterprise servers. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 235–242. ACM, 2010.

[5] 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*, 24(13):1397–1420, 2011.

[6] A. Beloglazov and R. Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *Parallel and Distributed Systems, IEEE Transactions on*, 24(7):1366–1379, 2013.

[7] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

[8] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, pages 119–128. IEEE, 2007.

[9] 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. *Softw., Pract. Exper.*, 2011.

[10] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. pages 381–398. Springer, 2003.

[11] K.-T. Chen, C. Chen, and P.-H. Wang. Network aware load-balancing via parallel vm migration for data centers. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*, pages 1–8. IEEE, 2014.

[12] L. Chen, H. Shen, and K. Sapra. Rial: Resource intensity aware load balancing in clouds. In *INFOCOM, 2014 Proceedings IEEE*, pages 1294–1302. IEEE, 2014.

[13] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira. Effective vm sizing in virtualized data centers. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 594–601, May 2011.

[14] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira. Effective vm sizing in virtualized data centers. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 594–601, May 2011.

[15] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[16] Q. Gao, P. Tang, T. Deng, and T. Wo. Virtualrank: A prediction based load balancing technique in virtual computing environment. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 247–256. IEEE, 2011.

[17] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 579–586, 1999.

[18] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. IEEE, 2010.

[19] Google cluster data. https://code.google.com/p/googleclusterdata/.

[20] J. Hu, J. Gu, G. Sun, and T. Zhao. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*, pages 89–96. IEEE, 2010.

[21] H. Jin, D. Pan, J. Xu, and N. Pissinou. Efficient vm placement with multiple deterministic and stochastic resources in data centers. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2505–2510, Dec 2012.

[22] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: Measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 202–208, New York, NY, USA, 2009. ACM.

[23] J. Li and H. Kameda. Load balancing problems for multiclass jobs in distributed/parallel computer systems. *Computers, IEEE Transactions on*, 47(3):322–332, Mar 1998.

[24] 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.

[25] 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.

[26] A. Sallam and K. Li. A multi-objective virtual machine migration policy in cloud systems. *The Computer Journal*, 57(2):195–204, 2014.

[27] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 559–570. IEEE, 2011.

[28] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 5. ACM, 2011.

[29] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee. Application-aware virtual machine migration in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 66–70. IEEE, 2011.

[30] A. Singh, M. Korupolu, and D. Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 53. IEEE Press, 2008.

[31] A. N. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *J. ACM*, 32(2):445–465, Apr. 1985.

[32] 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.

[33] P. Venkata Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, 13(5):2292–2303, 2013.

[34] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In M. Jaatun, G. Zhao, and C. Rong, editors, *Cloud Computing*, volume 5931 of *Lecture Notes in Computer Science*, pages 254–265. Springer Berlin Heidelberg, 2009.

[35] M. Wang, X. Meng, and L. Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 71–75, April 2011.

[36] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938, 2009.

[37] Z. Xiao, W. Song, and Q. Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1107–1117, June 2013.

[38] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li. iaware: Making live migration of virtual machines interference-aware in the cloud. *Computers, IEEE Transactions on*, PP(99):1–1, 2013.

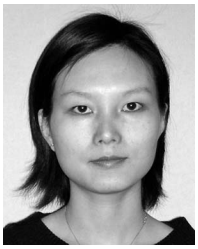
[39] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang. Live migration of multiple virtual machines with resource reservation in cloud computing environments. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 267–274. IEEE, 2011.

[40] Q. Zhu, J. Zhu, and G. Agrawal. Power-aware consolidation of scientific workflows in virtualized environments. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–12, Washington, DC, USA, 2010. IEEE Computer Society.

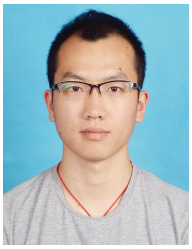
Lei Yu received his BS degree and MS degree in computer science from Harbin Institute of Technology, China. He is currently working towards the PhD degree in the Department of Computer Science at Georgia State University, USA. His research interests include cloud computing, sensor networks, wireless networks, and network security.

Liuhua Chen received both his BS degree and MS degree from Zhejiang University, in 2008 and 2011, and is currently working toward the PhD degree in the Department of Electrical and Computer Engineering at Clemson University. His research interests include distributed and parallel computer systems, and cloud computing. He is a student member of the IEEE.

Zhipeng Cai received his PhD and MS degrees from the Department of Computing Science at University of Alberta, and BS degree from the Department of Computer Science and Engineering at Beijing Institute of Technology. He is currently an Assistant Professor in the Department of Computer Science at Georgia State University. Prior to joining GSU, Dr. Cai was a research faculty in the School of Electrical and Computer Engineering at Georgia Institute of Technology. Dr. Cai's research areas focus on Networking and Big data. Dr. Cai is the recipient of an NSF CAREER Award.



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 Associate 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.



Yi Liang received his MS degree from the Department of of Computer Science at University of Science and Technology of China (USTC) in 2014, and is currently working towards the PhD degree in the Department of Computer Science at Georgia State University, USA. His research interests include cloud computing, big data and networking.

Yi Pan received the BEng and MEng degrees in computer engineering from Tsinghua University, China, and the PhD degree in computer science from the University of Pittsburgh, USA. He is the chair and a professor in the Department of Computer Science at Georgia State University, and a Changjiang chair professor in the Department of Computer Science at Central South University, China. His research interests include parallel and distributed computing, networks, and bioinformatics. He has published more than 100 journal papers with 50 papers published in various IEEE/ACM journals. In addition, he has published more than 100 papers in refereed conferences. He has authored/edited 34 books (including proceedings) and contributed many book chapters. He has served as the editor in chief or an editorial board member for 15 journals, including six IEEE Transactions. He is a senior member of the IEEE.