Burstiness-Aware Resource Reservation for Server Consolidation in Computing Clouds

Sheng Zhang, *Member, IEEE*, Zhuzhong Qian, *Member, IEEE*, Zhaoyi Luo, Jie Wu, *Fellow, IEEE*, and Sanglu Lu, *Member, IEEE*

Abstract—In computing clouds, burstiness of a virtual machine (VM) workload widely exists in real applications, where spikes usually occur aperiodically with low frequency and short duration. This could be effectively handled through dynamically scaling up/down in a virtualization-based computing cloud; however, to minimize energy consumption, VMs are often highly consolidated with the minimum number of physical machines (PMs) used. In this case, to meet the dynamic runtime resource demands of VMs in a PM, some VMs have to be migrated to some other PMs, which may cause potential performance degradation. In this paper, we investigate the burstiness-aware server consolidation problem from the perspective of resource reservation, i.e., reserving a certain amount of extra resources on each PM to avoid live migrations, and propose a novel server consolidation algorithm, QUEUE. We first model the resource requirement pattern of each VM as a two-state Markov chain to capture burstiness, then we design a resource reservation strategy for each PM based on the stationary distribution of a Markov chain. Finally, we present QUEUE, a complete server consolidation algorithm with a reasonable time complexity. We also show how to cope with heterogenous spikes and provide remarks on several extensions. Simulation and testbed results show that, QUEUE improves the consolidation ratio by up to 45 percent with large spike size and around 30 percent with normal spike size compared with the strategy that provisions for peak workload, and achieves a better balance between performance and energy consumption in comparison with other commonly-used consolidation algorithms.

Index Terms—Bursty workload, Markov chain, resource reservation, server consolidation, stationary distribution

1 Introduction

Cloud computing has been gaining more and more traction in the past few years, and it is changing the way we access and retrieve information [1]. The recent emergence of virtual desktop [2] has further elevated the importance of computing clouds. As a crucial technique in modern computing clouds, virtualization enables one physical machine (PM) to host many performance-isolated virtual machines (VMs). It greatly benefits a computing cloud where VMs running various applications are aggregated together to improve resource utilization. It has been shown in previous work [3] that, the cost of energy consumption, e.g., power supply, and cooling, occupies a significant fraction of the total operating costs in a cloud. Therefore, making optimal utilization of underlying resources to reduce the energy consumption is becoming an important issue [4], [5]. To cut back the energy consumption in clouds, server consolidation is proposed to tightly pack VMs to reduce the number of running PMs; however, VMs' performance may be seriously affected if VMs are not appropriately placed, especially in a highly consolidated cloud.

We observed that the variability and burstiness of VM workload widely exists in modern computing clouds, as evidenced in prior studies [4], [6], [7], [8], [9]. Take a typical web server for example, burstiness may be caused by flash crowed with bursty incoming requests. We all know that VMs should be provisioned with resources commensurate with their workload requirements [10], which becomes more complex when considering workload variation. As shown in Fig. 1, two kinds of resource provisioning strategies are commonly used to deal with workload burstiness—provisioning for peak workload and provisioning for normal workload. Provisioning for peak workload is favourable to VM performance guarantee, but it undermines the advantage of elasticity from virtualization and may lead to low resource utilization [1], [8], [9].

In contrast, provisioning for normal workload makes use of elasticity in cloud computing. In this case, to meet the dynamic resource requirements of VMs, local resizing and live migration are the two pervasively-used methods. Local resizing adaptively adjusts VM configuration according to the real-time resource requirement with negligible time and computing overheads [11]. On the other hand, live migration moves some VM(s) to a relatively idle PM, when local resizing is not able to allocate enough resources. However, in a highly consolidated computing cloud where resource contention is generally prominent among VMs, live migration may cause significant service downtime; furthermore, it also incurs noticeable CPU usage on the host PM [12], which probably degrades the co-located VMs' performance.

- *S. Zhang, Z. Qian, and S. Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: {sheng, qzz, sanglu}@nju.edu.cn.*
- *Z. Luo is with the David Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L3G1, Canada. E-mail: zhaoyi.luo@uwaterloo.ca.*
- *J. Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122. E-mail: jiewu@temple.edu.*

Manuscript received 3 Jan. 2015; revised 7 Apr. 2015; accepted 15 Apr. 2015. Date of publication 21 Apr. 2015; date of current version 16 Mar. 2016. Recommended for acceptance by X. Gu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2425403

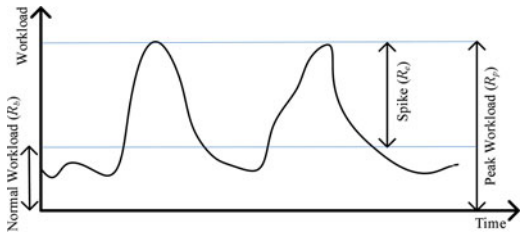


Fig. 1. An example of workload with bursty spikes.

In this paper, we propose to reserve some extra resources on each PM to accommodate bursty workload [13]. In doing so, when a resource spike occurs, VMs can be quickly reconfigured to the new level of resource requirement through local resizing with minimal overheads, instead of being migrated to some other PMs. Hence, the number of live migrations could be reduced considerably and the overall performance of a computing cloud could be improved.

Specifically, we investigate the problem of minimizing the amount of extra resources reserved on each PM during server consolidation while the overall performance is probabilistically guaranteed. By "probabilistically guaranteed", we mean that, the fraction of time within which the aggregated workloads of a PM exceed its physical capacity is not larger than a threshold. Imposing such a threshold rather than conducting live migration upon PM's capacity overflow is a way to tolerate minor fluctuations of resource usage (like the case of CPU usage) and to break the tradeoff between utilization and performance. Then, our problem can be formulated as an optimization, wherein the goal is to minimize the amount of resource reserved on each PM, and the constraint is that the capacity violation ratio of every PM is not larger than a predetermined threshold.

We use a two-state Markov chain to capture the burstiness of workload [7], and also shows how to learn the chain parameters. Inspired by the serving windows in queueing theory [14], we abstract the resources reserved on each PM for workload spikes as blocks. Denoting by $\theta(t)$ the number of busy blocks at time t on a PM, we show that a sequence of $\theta(0), \theta(1), \theta(2), \ldots$ has the Markov property, namely that, the next state only depends on the current state and not on the past sequence of states. Then we develop a novel server consolidation algorithm, QUEUE, based on the stationary distribution of this Markov chain. We also show how to further improve the effectiveness of QUEUE with more careful treatment of heterogenous workload spikes. Simulation and testbed results show that, QUEUE improves the consolidation ratio by up to 45 percent with large spike size and around 30 percent with normal spike size compared with the strategy that provisions for peak workload, and achieves a better balance between performance and energy consumption in comparison with other commonly-used consolidation algorithms. The contributions of our paper are three-fold.

1) To the best of our knowledge, we are the first to quantify the amount of reserved resources with consideration of workload burstiness. We propose to use the two-state Markov chain model to capture workload burstiness, and we present a formal problem description and its NP-completeness.
2) We develop a novel algorithm, QUEUE, for burstiness-aware resource reservation, based on the stationary distribution of a Markov chain. We also show how to cope with heterogeneous spikes to further improve the performance of QUEUE.
3) Extensive simulations and testbed experiments are conducted to validate the effectiveness and advantages of QUEUE.

We now continue by presenting related work in Section 2 and our model in Section 3. Problem formulation is provided in Section 4. We show the details of QUEUE in Section 5. Heterogeneous spikes are handled in Section 6. Evaluation results are presented in Section 7. Before concluding the paper in Section 9, we discuss known issues and extensions of QUEUE in Section 8.

2 Related Work

Most of prior studies [3], [15], [16] on server consolidation focused on minimizing the number of active PMs from the perspective of bin packing (BP). A heterogeneity-aware resource management system for dynamic capacity provisioning in clouds was developed in [17]. Stable resource allocation in geographically-distributed clouds was considered in [18]. Network-aware virtual machine placement was considered in [19]. Scalable virtual network models were designed in [8], [20] to allow cloud tenants to explicitly specify computing and networking requirements to achieve predictable performance.

In a computing cloud, burstiness of workload widely exists in real applications, which becomes an inevitable characteristic in server consolidation [1], [4], [6], [7], [21]. Some recent works [22], [23] used stochastic bin-packing (SBP) techniques to deal with variable workloads, where workload is modeled as random variable. Some other research [10], [24], [25] studied the SBP problem assuming VM workload follows normal distribution. Several other studies [26], [27] focused on workload prediction while the application runs. Different from them, in our model a lower limit of provisioning is set at the normal workload level which effectively prevents VM interference caused by unpredictable behaviors from co-located VMs.

Markov chain was used to inject burstiness into a traditional benchmark in [7]. Several works [5], [28], [29] studied modeling and dynamic provisioning of bursty workload in cloud computing. A previous study [30] proposed to reserve a constant level of hardware resource on each PM to tolerate workload fluctuation; but how much resource should be reserved was not given. To the best of our knowledge, we are the first to quantify the amount of reserved resources with consideration on various, but distinct, workload burstiness.

3 Modeling Virtual Machine Workload

3.1 Two-State Markov Chain

It has been well recognized in previous studies [4], [6], [7] that VM workload is time-varying with bursty spikes, as shown in Fig. 1. Several works [9], [10], [22], [23], [24], [25] modeled the workload of a VM as a random variable,

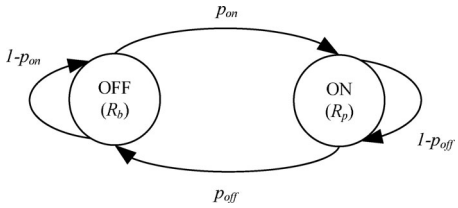


Fig. 2. Two-state Markov chain. The "ON" state represents peak workload (R_p) while the "OFF" state represents normal workload (R_b). p_{on} and p_{off} are the state switch probabilities.

which follows the Bernoulli distribution in [9] or normal distribution in [10], [24], [25]. Different from these works, we model the workload of a VM as a two-state Markov chain, which takes the additional dimension of time into consideration, and thus describes the characteristics of spikes more precisely.

Fig. 2 shows an example. We denote the resource requirements of peak workload, normal workload, and workload spike by R_p, R_b, and R_e, respectively, where $R_e = R_p - R_b$ as demonstrated in Fig. 1. The "ON" state represents peak workload while the "OFF" state represents normal workload. We use p_{on} and p_{off} to denote the state switch probabilities. More specifically, if a VM is in the ON state, then the probability of it switching to OFF at the next time is p_{off}, and remaining ON is $1 - p_{off}$. Similarly if a VM is in the OFF state, then the probability of it switching to ON at next time is p_{on} and remaining ON is $1 - p_{on}$. We emphasize that this model is able to describe the characteristics of spikes precisely—intuitively, R_e denotes the size of a spike, and p_{on} denotes the frequency of spike occurrence. Thus, each VM can be described by a four-tuple

$$V_i = \left(p^i_{on}, p^i_{off}, R^i_b, R^i_e\right), \forall 1 \le i \le n, \tag{1}$$

where n is the number of VMs.

3.2 Learning Model Parameters

This section provides a simple strategy for cloud tenants to generate model parameters for their VM workload. It consists of two phases.

First, a cloud tenant must have the workload traces and guarantees that they will be consistent with the realistic deployment in computing clouds. This could be achieved by tentatively deploying VMs in a cloud for a relatively short period; the cloud system collects the resource usage traces and feeds them back to tenants.

Second, given a VM workload trace, a cloud tenant generates a four-tuple. We use Fig. 3 as an illustration, where the solid black curve represents the workload over time. Given the predetermined R_b and R_e, we conservatively round the solid black curve up to the dashed red curve. Denote by $W(t)$ the workload at time t in the dashed red curve, *e.g.*, $W(4) = R_p$, and $W(7) = R_b$.

Denote by S_{FF} the number of switches from state "OFF" to state "OFF" in two consecutive time slots during the time period of interest; denote by S_{FN} the number of switches from state "OFF" to state "ON" in two consecutive time slots during the time period of interest, e.g., $S_{FF} = 5$ and $S_{FN} = 2$ in Fig. 3. Similarly, we can define S_{NN} and S_{NF},

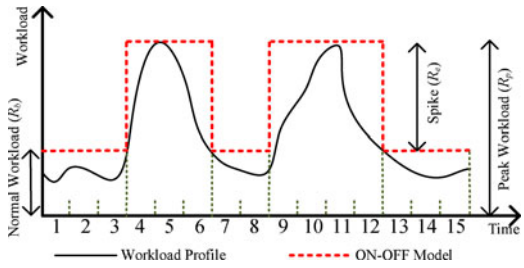


Fig. 3. Given the predetermined R_b and R_e, we conservatively round the solid black curve up to the dashed red curve, based on which we can calculate p_{on} and p_{off}.

which are equal to 5 and 2, respectively, in the figure. It is then easy to see that

$$p_{on} = \frac{S_{FN}}{S_{FN} + S_{FF}}, \text{ and } p_{off} = \frac{S_{NF}}{S_{NF} + S_{NN}}.$$

3.3 Potential Benefits

The two-state Markov chain model allows cloud tenants to flexibly control the tradeoff between VM performance and deployment cost through adjusting R_b and R_e.

When a tenant wants to maximize VM performance, the tenant should choose a large R_b and a small R_e. As we will show later in this paper, there may be multiple workload spikes that share some common physical resources. Thus, when the aggregated amount of workload spikes that simultaneously occur is larger than the amount of the shared common resources, capacity overflow happens and VM performance is probably affected.

When a tenant wants to minimize deployment cost, the tenant should choose a small R_b and a large R_e. By "deployment cost", we mean the fee which is paid by a cloud tenant to a cloud provider. Since physical resources are opportunistically shared among multiple workload spikes, the charge for workload spike should be smaller than that for normal workload [9]. Therefore, decreasing R_b helps tenants to reduce the deployment cost.

Our model is also a tradeoff between modeling complexity and precision. We could model time-varying workload by three-state or even more states of Markov chain, which should capture the workload bustiness more precisely; however, the complexity in learning model parameters and allocating physical resources increases as well, which may complicate the interactions between cloud providers and tenants.

4 Problem Formulation

We consider a computing cloud with one-dimensional resource; for scenarios with multi-dimensional resources, we provide a few remarks in Section 8. There are m physical machines in the computing cloud, and each PM is described by its physical capacity

$$H_j = (C_j), \forall 1 \le j \le m. \tag{2}$$

We use a binary matrix $\mathbf{X} = [x_{ij}]_{n \times m}$ to represent the results of placing n VMs on m PMs: $x_{ij} = 1$, if V_i is placed on H_j, and 0 otherwise. We assume that the workloads of VMs are mutually independent. Let $W_i(t)$ be the resource

Symbol	Meaning
n	the number of VMs
V_i	the i-th VM
R_b^i	the normal workload size of V_i
R_e^i	the spiky workload size of V_i
R_p^i	the peak workload size of V_i, and $R_p^i = R_b^i + R_e^i$
p_{on}^i	the probability of V_i switching from OFF to ON
p_{off}^i	the probability of V_i switching from ON to OFF
m	the number of PMs
H_j	the j-th PM
C_j	the physical capacity of H_j
x_{ij}	the variable indicating whether VM_i is placed on PM_j
Φ_j	the capacity overflow ratio of PM PM_j
ρ	the threshold of capacity overflow ratio

Fig. 4. Main notations for quick reference.

requirements of V_i at time t. According the Markov chain model, we have

$$W_i(t) = \begin{cases} R_b^i & \text{if } V_i \text{ is in the "OFF" state at time } t, \\ R_p^i & \text{if } V_i \text{ is in the "ON" state at time } t. \end{cases}$$

Then, the aggregated resource requirement of VMs on PM H_j is $\sum_{i=1}^{n} x_{ij} W_i(t)$.

Let CO_j^t indicate whether the capacity overflow happens on PM H_j at time t, i.e.,

$$CO_j^t = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_{ij} W_i(t) > C_j, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, the results of VM placement should guarantee that the capacity constraint is satisfied on each PM at the beginning of the time period of interest, i.e.,

$$CO_j^0 = 0, \forall 1 \le j \le m.$$

We now can define our metric for probabilistic performance guarantee—*capacity overflow ratio* (COR), which is the fraction of time that the aggregated workloads of a PM exceed its physical capacity. Denoting the capacity overflow ratio of PM H_j as Φ_j, we have

$$\Phi_j = \frac{\sum_{1 \le t \le T} CO_j^t}{T},$$

where T is the length of the time period of interest. It is easy to see that, a smaller Φ_j implies a better performance of H_j. The performance of each PM should be probabilistically guaranteed, so we have

$$\Phi_j \le \rho, \forall 1 \le j \le m. \tag{3}$$

Here, ρ is a predetermined value serving as the threshold of COR. Main notations are summarized in Fig. 4 for quick reference. Our problem can be stated as follows.

Problem 1 (Burtiness-Aware Server Consolidation, BASC). *Given a set of n VMs and a set of m PMs, find a VM-to-PM mapping $\mathbf{X}$ to minimize the number of PMs used while making sure that (1) the initial placement satisfies capacity constraint, and (2) the capacity overflow ratio of each PM is not larger than the threshold ρ. It can be formally formulated as follows:*

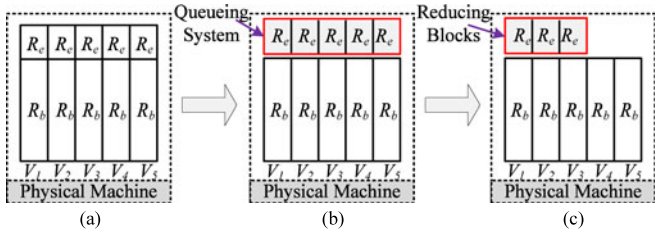


Fig. 5. An illustration of the evolution process. (a) The original provisioning strategy for peak workload. (b) Gathering all R_e's together to form a queueing system. (c) Reducing the number of blocks while still satisfying Eq. (3).

$$\begin{aligned} \min \quad & \left| \left\{ j \middle| \sum_{i=1}^{n} x_{ij} > 0, 1 \le j \le m \right\} \right| \\ s.t. \quad & CO_j^0 = 0, \forall 1 \le j \le m \\ & \Phi_j \le \rho, \forall 1 \le j \le m. \end{aligned} \tag{4}$$

Here, $|\mathcal{S}|$ denotes the cardinality of set $\mathcal{S}$. In the following theorem, we can prove that, the BASC problem is NP-complete.

Theorem 1. *The BASC problem is NP-complete.*

Proof. We prove this theorem by reduction from the Bin Packing problem [31], which is NP-complete. The decision version of the BP problem is as follows. Given n items with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$, can we pack them in no more than k unit-sized bins?

Given an instance of the decision version of the BP problem, we can construct an instance of the decision version of our problem as follows: let $R_b^i = R_p^i = s_i$, $\forall 1 \le i \le n$; let $m = k$; let $C_j = 1$, $\forall 1 \le j \le m$; and let $\rho = 0$, i.e., capacity overflow is not allowed.

It is not hard to see that the construction can be finished in polynomial time; thus, we reduce solving the NP-complete BP problem to solving a special case of our problem, implying that our problem is NP-hard. It is easy to verify that the BASC problem is also in NP; the theorem follows immediately. □

5 Burstiness-Aware Resource Reservation

In this section, we first present the main idea of our solution to the BASC problem, then we design a resource reservation strategy for a single PM, based on which we develop QUEUE, a complete server consolidation algorithm. In the end, we provide a concrete example to help readers better understand our algorithm.

5.1 Overview of QUEUE

We propose reserving a certain amount of physical resources on each PM to accommodate workload spikes. The main idea is to abstract the reserved spaces as blocks (i.e., serving windows in queueing theory). We give an informal illustration of the evolution process of our queueing system in Fig. 5.

Initially, all VMs are provisioned by $R_b + R_e$, and each VM has its own block (denoted as R_e in Fig. 5). A VM uses only its R_b part during periods of normal workload, however, when a workload spike occurs, the extra R_e part is put

into use. We note that, the collected R_e's altogether form a queueing system—when a workload spike occurs in a VM, the VM enters the queueing system and occupies one of the idle blocks; when the spike disappears, the corresponding VM leaves the queueing system and releases the block. It is worth noting that, there is no waiting space in the queueing system; thus, the PM capacity constraint would be violated if a workload spike occurs while all the blocks are occupied, which never happens when the number of blocks equals the number of co-located VMs (as shown in Fig. 5b). However, we may find that a certain number of blocks are idle for the majority of the time in Fig. 5b, so we can reduce the number of blocks while only incurring very few capacity violations (as shown in Fig. 5c). Therefore, our goal becomes reserving minimal number of blocks on each PM while the performance constraint in Eq. (3) is still satisfied.

5.2 Resource Reservation Strategy for a Single PM

In this section, We focus on resource reservation for a single PM. For the sake of convenience, we set the size of each block as the size of the maximum spike of all co-located VMs on a PM. In Section 6, we will present how to cope with heterogenous workload spikes in an effort to further improve the performance of QUEUE. We also assume that all VMs have the same state switch probabilities, *i.e.*, $p_{on}^i = p_{on}$ and $p_{off}^i = p_{off}$, for all $1 \le i \le n$. In Section 8, we will show how to cluster VMs when they have different state switch probabilities.

Suppose there are k VMs on the PM of interest and initially each VM V_i occupies R_b^i resources. We initialize the number of blocks reserved on this PM as k, and our objective is to reduce the number of blocks to K ($K < k$), while the capacity overflow ratio Φ does not exceed the threshold ρ. Let $\theta(t)$ be the number of busy blocks at time t, implying that, there are $\theta(t)$ VMs in the ON state and $(k - \theta(t))$ VMs in the OFF state. Let $O(t)$ and $I(t)$ denote the number of VMs that switch state from ON to OFF (i.e., VMs that leave the queueing system) and from OFF to ON (i.e., VMs that enter the queueing system) at time t, respectively. We use $\binom{x}{y}$ to denote $\frac{x!}{y!(x-y)!}$, i.e., x choose y. Since the workloads of VMs are mutually independent, we have

$$Pr\{O(t) = x\} = \binom{\theta(t)}{x} p_{off}^x (1 - p_{off})^{\theta(t)-x},$$

and

$$Pr\{I(t) = x\} = \binom{k - \theta(t)}{x} p_{on}^x (1 - p_{on})^{k-\theta(t)-x},$$

which suggest that both $O(t)$ and $I(t)$ follow the binomial distribution:

$$\begin{cases} O(t) \sim B(\theta(t), p_{off}), \\ I(t) \sim B(k - \theta(t), p_{on}). \end{cases} \tag{5}$$

Without loss of generality, we assume that the switch between two consecutive states of all VMs happens at the end of each time interval. Then we have the recursive relation of $\theta(t)$,

$$\theta(t+1) = \theta(t) - O(t) + I(t). \tag{6}$$

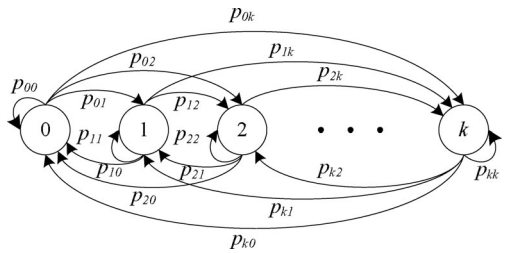


Fig. 6. The transition graph of the stochastic process $\{\theta(0), \theta(1), \ldots, \theta(t), \ldots\}$. The stochastic process is said to be in state i ($1 \le i \le k$) if the number of busy blocks is i. p_{ij} is the transition probability from state i to state j.

Combining Eqs. (5) and (6) together, we see that, the next state $\theta(t+1)$ only depends on the current state $\theta(t)$ and not on the past sequence of states $\theta(t-1)$, $\theta(t-2), \ldots, \theta(0)$. Therefore, the stochastic process $\theta(0)$, $\theta(1), \ldots$ of discrete time ($\{0, 1, 2, \ldots\}$) and discrete space ($\{0, 1, 2, \ldots, k\}$) is a Markov chain. The stochastic process is said to be in state i ($1 \le i \le k$) if the number of busy blocks is i. Fig. 6 shows the transition graph of the chain.

Let p_{ij} be the transition probability from state i to state j. That is to say, if $\theta(t) = i$, then the probability that $\theta(t+1) = j$ is p_{ij}. For the sake of convenience, when $y > x$ or $y \le x < 0$, we let $\binom{x}{y}$ be 0. Then, p_{ij} can be derived as follows:

$$\begin{aligned} p_{ij} =& Pr\{\theta(t+1) = j | \theta(t) = i\} \\ =& \sum_{r=0}^{i} Pr\{O(t) = r, I(t) = j - i + r | \theta(t) = i\} \\ =& \sum_{r=0}^{i} Pr\{O(t) = r | \theta(t) = i\} \\ & \times Pr\{I(t) = j - i + r | \theta(t) = i\} \\ =& \sum_{r=0}^{i} \binom{i}{r} p_{off}^r (1 - p_{off})^{i-r} \\ & \times \binom{k-i}{j-i+r} p_{on}^{j-i+r} (1 - p_{on})^{k-j-r}. \end{aligned} \tag{7}$$

In the above formula, the first and second equations are due to the definition of p_{ij} and the recursive relation of $\theta(t)$, respectively. Observing that $O(t)$ and $I(t)$ are independent of each other, we get the third equation. The last equation can be obtained by replacing $O(t)$ and $I(t)$ with their distributions.

The stochastic matrix $\mathbf{P} = [p_{ij}]_{(k+1)\times(k+1)}$ is called the transition matrix of the Markov chain. We see that, it does not depend on the time. Let $\pi^{(t)} = (\pi_0^{(t)}, \pi_1^{(t)}, \ldots, \pi_k^{(t)})$ be the distribution of the chain at time t, i.e., $\pi_h^{(t)} = Pr\{\theta(t) = h\}$, $\forall 0 \le h \le k$. For our chain, which is finite, $\pi^{(t)}$ is a vector of $k+1$ nonnegative entries such that $\sum_{h=0}^{k} \pi_h^{(t)} = 1$. In linear algebra, vectors of this type are called stochastic vectors. Then, it holds that

$$\pi^{(t+1)} = \pi^{(t)} \mathbf{P}.$$

Suppose π is a distribution over the state space $\{0, 1, 2, \ldots, k\}$ such that, if the chain starts with an initial

distribution $\pi^{(0)}$ that is equal to π, then after a transition, the distribution of the chain is still $\pi^{(1)} = \pi$. Then the chain will stay in the distribution π forever:

$$\pi \xrightarrow{P} \pi \xrightarrow{P} \pi \xrightarrow{P} \cdots\cdots$$

Such π is called a stationary distribution. For our chain, we have the following theorem.

Theorem 2. *For the Markov chain defined in Fig. 6 and Eq. (7), given an arbitrary initial distribution $\pi^{(0)}$, $\pi^{(t)}$ will converge to the same distribution π, which satisify*

$$\pi = \pi\mathbf{P}, \text{ and } \lim_{t\to\infty}(\pi^{(0)}\mathbf{P}^t)_h = \pi_h, \forall 0 \le h \le k.$$

Proof. According to the Markov chain convergence theorem [14], it is sufficient to prove that, our chain is finite, aperiodic, and irreducible. Since the number of VMs that a single PM can host is finite (in our case, it is k), the state space of our chain is finite. As shown in Eq. (7), $p_{ii} > 0$ for any state i, so all states are aperiodic. Finally, from any state i, we can reach any other state j, so the transition graph is strongly connected, implying that our chain is irreducible. The theorem follows immediately. □

When the chain stays in the stationary distribution, we see that π_h is equivalent to the proportion of times that the stochastic process is in state h. In our case, it means that π_h denotes the proportion of time wherein the number of busy resource blocks is h.

We then can derive the minimum number of blocks that keeps the capacity overflow ratio not larger than ρ. Denote by K the minimum number of blocks; we argue that K satisfies the following constraint:

$$\sum_{h=0}^{K-1}\pi_h < 1-\rho \le \sum_{h=0}^{K}\pi_h, \tag{8}$$

which suggests that K is the minimum number that guarantees $\sum_{h=0}^{K}\pi_h \ge 1-\rho$.

This is because, when he number of reserved blocks on PM H_j is reduced from k to K, if the queueing system is in state h, which is larger than K, then capacity overflow occurs, i.e., $CO_j^t = 1$, and vice versa. Thus we have

$$\Phi_j = \frac{\sum_{1\le t\le T} CO_j^t}{T} = \sum_{h=K+1}^{k}\pi_h = 1 - \sum_{h=0}^{K}\pi_h \le \rho.$$

We now show how to calculate the stationary distribution π. According to its definition, we have $\pi = \pi\mathbf{P}$, which is equivalent to the following homogeneous system of linear equations that can be solved by Gaussian elimination

$$\begin{cases} \sum_{h=0}^{k}\pi_h p_{h0} - \pi_0 = 0 \\ \sum_{h=1}^{k}\pi_h p_{h1} - \pi_1 = 0 \\ \qquad\cdots\cdots \\ \sum_{h=k}^{k}\pi_h p_{hk} - \pi_k = 0. \end{cases} \tag{9}$$

Algorithm 1 summarizes the entire process of how to calculate the minimum number of reserved blocks, given parameters k, p_{on}, p_{off}, and ρ. Calculating the transition matrix $\mathbf{P}$ requires $O(k^3)$ time; solving the linear equations using Gaussian elimination costs roughly $O(k^3)$ time; finding K that satisfies Eq. (8) needs $O(k)$ time. Therefore, the time complexity of Algorithm 1 is $O(k^3)$.

Algorithm 1. Calculating Minimum Blocks (CalMinBlk)

Input: k, the number of co-located VMs on a PM;
p_{on}, the switch probability from "OFF" to "ON";
p_{off}, the switch probability from "ON" to "OFF";
ρ, capacity overflow ratio threshold
Output: K, the minimum number of blocks that should be reserved on a PM
1: Calculate the transition matrix $\mathbf{P}$ using Eq. (7)
2: Prepare the coefficient matrix of the homogeneous system of linear equations described in Eq. (9)
3: Solve the the homogeneous system via Gaussian elimination and get the stationary distribution π
4: Calculate K from π using Eq. (8)
5: **return** K;

5.3 QUEUE

In this section, we present the complete server consolidation algorithm, QUEUE, which is to place a set of n VMs onto a set of m PMs.

As we mentioned before, we conservatively set the size of a single block as the size of the maximum spike of all the VMs on each PM, which may result in low utilization if the workload spikes of the co-located VMs differ too much. Therefore, in QUEUE, we tend to place VMs with similar R_e's on the same PM in an effort to reduce the average size of a single block. Algorithm 2 shows the details of QUEUE, which consists of three phases.

Algorithm 2. QUEUE

Input: $V_1, V_2, \ldots, V_n$, specifications of n VMs;
$H_1, H_2, \ldots, H_m$, specifications of m PMs;
p_{on}, the switch probability from "OFF" to "ON";
p_{off}, the switch probability from "ON" to "OFF";
ρ, capacity overflow ratio threshold;
d, the maximum number of VMs allowed on a PM;
Output: $\mathbf{X}$, a VM-to-PM placement matrix
1: // Preprocessing phase
2: $MinN \leftarrow$ an array of size $d+1$
3: $MinN[0] \leftarrow 0$
4: **for** each $k \in [1, d]$ **do**
5: $\quad MinN[k] \leftarrow$ CalMinBlk(k, p_{on}, p_{off}, ρ)
6: **end for**
7: // Sorting phase
8: Cluster VMs based on their R_e's
9: Sort clusters in descending order of R_e
10: In each cluster, sort VMs in descending order of R_b
11: Sort PMs in descending order of C
12: // FFD-based placement phase
13: $\mathbf{X} \leftarrow [0]_{n\times m}$
14: **for** each V_i in the sorted order **do**
15: $\quad$ place V_i on the first H_j (in the sorted order) that satisfies Eq. (10), and set $x_{ij} \leftarrow 1$;
16: **end for**
17: **return** $\mathbf{X}$;

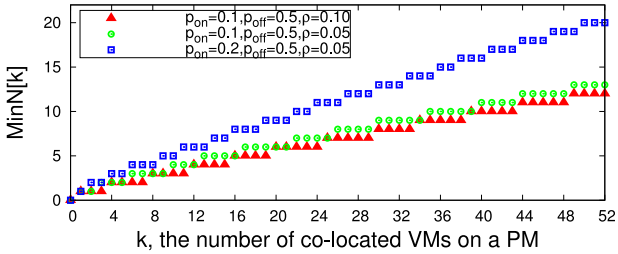


Fig. 7. The minimum number, $MinN[k]$, of blocks that should be reserved on a PM, given different settings of p_{on}, p_{off} and ρ.

In the preprocessing phase (lines 1-6), we introduce an array named $MinN$, which stores the information about the minimum number of blocks that need to be reserved on a PM, given k, p_{on}, p_{off}, and ρ. That is, if there are k VMs placed on a PM, then we need to reserve $MinN[k]$ blocks to bound the capacity overflow ratio. Without loss of generality, we assume that a single PM can host up to d VMs, thus we can calculate the array $MinN[k]$ for all possible $k \in [1, d]$ before VM placement. We also let $MinN[0] = 0$ for compatibility. Fig. 7 shows the array $MinN$ given different settings of p_{on}, p_{off}, and ρ. We notice that, for the same k, when the capacity overflow ratio threshold ρ increases (from green circles to red triangles), $MinN[k]$ decreases; when p_{on} increases (from green circles to blue squares), $MinN[k]$ increases. These observations are consistent with our intuitions.

During the sorting phase (lines 7-11), we first cluster all VMs so that VMs with similar R_e's are in the same cluster,[1] and then sort these clusters in the descending order of R_e. In each cluster, we sort VMs in the descending order of R_b. We also sorted PMs in the descending order of the physical capacity C. This is a cluster-level heuristic to let co-located VMs have similar R_e's, thus to minimize the average size of blocks on all PMs.

In the third phase (lines 12-16), we adopt the First Fit Decrease (FFD) [31] heuristic to place VMs on PMs. For each V_i in the sorted order, we place V_i on the first PM H_j that satisfies the following constraint:

$$\max\{R_e^i, \max\{R_e^s | s \in T_j\}\} \times MinN[|T_j| + 1] + R_b^i + \sum_{s \in T_j} R_b^s \leq C_j, \tag{10}$$

where T_j denotes the set of indices of VMs that have already been placed on H_j, and C_j is the capacity of H_j. We note that, the size of the reserved resources is the block size multiplying the number of blocks, where block size is conservatively set to the maximum R_e among all co-located VMs. Therefore, this constraint indicates that, VM V_i can be placed on H_j *if and only if* the sum of the new size of the queueing system and the new total size of R_b's does not exceed the physical capacity of H_j. If Eq. (10) holds, we set x_{ij} be 1. At the end of QUEUE, we return the VM-to-PM mapping result, i.e., $\mathbf{X}$.

Finally, we present the complexity analysis of QUEUE. In the preprocessing phase, Algorithm 1 is invoked at most $d+1$ times. Remember that the time complexity of Algorithm 1 is $O(k^3)$, thus, this phase costs $O(d^4)$ time. The simple clustering phase takes $O(n)$ time. More developed clustering techniques are out of the scope of this paper. The sorting phase takes $O(n \log n)$ time. The FFD-based placement phase takes $O(mn)$ time. Overall, the time complexity of the complete consolidation algorithm is $O(d^4 + n \log n + mn)$.

1. We first use linear time to find the maximum, $maxR$, and minimum, $minR$, of n R_e's, then partition all R_e's into c clusters, where the ith cluster contains those R_e's that satisfy $minR + \frac{i-1}{c}(maxR - minR) \leq R_e < minR + \frac{i}{c}(maxR - minR)$, for $i = 1, 2, \ldots, c$.

5.4 A Concrete Example

In this section, we provide a concrete example to better explain the details of QUEUE. In our example, there are $n = 8$ VMs and $m = 3$ PMs with the following parameters (see Eqs. (1) and (2)):

$$V_1 = (0.1, 0.5, 15, 13), V_2 = (0.1, 0.5, 15, 13),$$
$$V_3 = (0.1, 0.5, 20, 15), V_4 = (0.1, 0.5, 20, 10),$$
$$V_5 = (0.1, 0.5, 25, 15), V_6 = (0.1, 0.5, 10, 9),$$
$$V_7 = (0.1, 0.5, 15, 10), V_8 = (0.1, 0.5, 10, 9),$$

and

$$H_1 = H_2 = H_3 = (100).$$

As we mentioned in Section 5.2, we assume that all VMs have the same state switch probabilities. The threshold ρ of capacity overflow ratio is set to 0.05. The maximum number d of VMs allowed on a single PM is 4. We now present the details of running QUEUE.

In the preprocessing phase, we are to generate the array $MinN$. Taking $k = 4$ for example, according to Eq. (7), we can obtain the transition matrix $\mathbf{P}$:

$$\mathbf{P} = \begin{bmatrix} 0.6561 & 0.2916 & 0.0486 & 0.0036 & 0.0001 \\ 0.3645 & 0.4860 & 0.1350 & 0.0140 & 0.0005 \\ 0.2025 & 0.4500 & 0.2950 & 0.0500 & 0.0025 \\ 0.1125 & 0.3500 & 0.3750 & 0.1500 & 0.0125 \\ 0.0625 & 0.2500 & 0.3750 & 0.2500 & 0.0625 \end{bmatrix}.$$

By solving $\pi = \pi\mathbf{P}$, we have the stationary distribution:

$$\pi = (0.4823, 0.3858, 0.1157, 0.0154, 0.0008).$$

Since $\pi_0 + \pi_1 < 1 - \rho \leq \pi_0 + \pi_1 + \pi_2$, we have $K = 2$, which is also the value of $MinN[4]$. Similarly, we can find that, $MinN[0] = 0$, $MinN[1] = MinN[2] = 1$, and $MinN[3] = 2$ (see the green circles in Fig. 7).

In the clustering phase, these eight VMs are first clustered into two groups, i.e., V_1, V_2, V_3, and V_5 are the first group, and the rest is the second group. In each cluster, we sort VMs to be in the descending order of their R_b's. The final order of VMs is V_5, V_3, V_1, V_2, V_4, V_7, V_6, and V_8.

In the third phase of QUEUE, we first try to place V_5 on H_1. Since $R_b^5 + R_e^5 < C_1$, it succeeds and we set $x_{51} = 1$. We then try to place V_3 on H_1. According to Equ. (10), we have

$$\max\{R_e^3, \max\{R_e^5\}\} \times MinN[|\{3\}| + 1] + R_b^3 + R_b^5 < C_1,$$

so we set $x_{31} = 1$. We then try to place V_1 on C_1, which also succeeds and we set $x_{11} = 1$. Similarly, we then find that, in the final placement, V_5, V_3, V_1, and V_6 are placed on H_1, and the rest is placed on H_2.

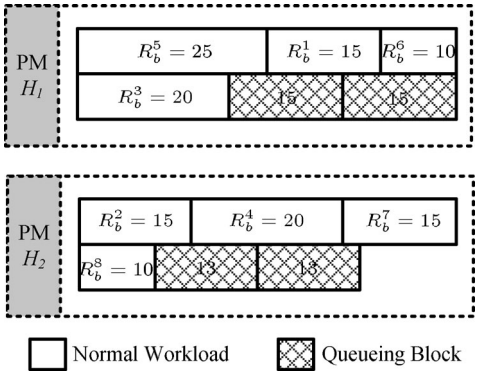


Fig. 8. The final placement of our example using QUEUE, where only two PMs are needed. We conservatively round the block size up to the maximum spike size of all co-located VMs on a PM, e.g., on PM H_1, the size of each queueing block is $\max\{R_e^5, R_e^3, R_e^1, R_e^6\} = 15$.

Fig. 8 shows the final placement of our example using QUEUE. Note that, we conservatively round the block size up to the maximum spike size of all co-located VMs on a PM, e.g., on PM H_1, the size of each queueing block is $\max\{R_e^5, R_e^3, R_e^1, R_e^6\} = 15$.

In contrast, without opportunistic resource sharing in the queueing blocks, if resources are provisioned for peak workload, then three PMs are needed to host these eight VMs, i.e., V_5, V_3, and V_6 are on H_1; V_1, V_2, and V_4 are on H_2; and the remaining two VMs are on H_3.

6 Coping with Heterogenous Spikes

In this section, we present how to improve the consolidation performance of QUEUE through more careful treatment of heterogenous workload spikes.

Remember that, in the last section, we conservatively round the block size up to the maximum spike size of all co-located VMs on each PM, as shown in Fig. 8. It is easy to see that, this kind of rounding may waste some physical resources. Let us take PM H_1 in Fig. 8 for example, by the current design of QUEUE, a total of 30 units of physical resources need to be reserved for workload spikes; however, instead of considering the four VMs (i.e., V_5, V_3, V_1, and V_6) together, we can partition them into two groups and consider each of them separately. For instance, we choose to have V_5 and V_3 in the first group, and have V_1 and V_6 in the second group. For the former group, since $MinN[2] = 1$, we only have to reserve one block with a size of $\max\{R_e^5, R_e^3\} = 15$; for the latter group, we also have to reserve one block with a size of $\max\{R_e^1, R_e^6\} = 13$. In doing so, a total of 28 units of resources are reserved, which is less than that in the previous case.

We, therefore, have the following intuition: on each PM, we can try to partition the co-located VMs into several groups and consider them separately, so as to improve QUEUE by reducing the amount of resources reserved for workload spikes.

A key problem in achieving our goal is how to partition a set of k VMs into non-overlapped groups, i.e., how to partition an integer k, which is an interesting and important problem in number theory [32]. A g-partition of an integer k is a multi-set $\{x_1, x_2, \ldots, x_i, \ldots, x_g\}$ with $x_i \geq 1$ for every element x_i and $x_1 + x_2 + \cdots + x_g = k$. For example, $\{1, 2, 4\}$ and $\{1, 3, 3\}$ are two possible three-partitions of integer 7. Denote by $p_g(k)$ the number of different g-partitions of an integer k, and by $p(k)$ the number of all possible partitions of an integer k. For example, $p_3(7) = 4$, $p(7) = 15$. According to [33], we have the following recursive relations of $p_g(k)$:

$$p_g(k) = p_{g-1}(k-1) + p_g(k-g). \tag{11}$$

Algorithm 3 shows the main steps to find the optimal ordered partition. Without loss of generality, we assume that, $R_e^1, R_e^2, \ldots, R_e^k$ are sorted in descending order of their values. The array $MinN$ is computed using Algorithm 1. Since the number of all possible partitions of an integer k is very large ($\log p(k) = O(\sqrt{k})$ [33]), enumerating them would be time-consuming. So we use G to restrict the maximum number of groups.

Algorithm 3. Finding Optimal Partition (FidOptPat)

Input: $MinN$, an array that stores the minimum numbers of blocks that need to be reserved on a PM, with respect to k, p_{on}, p_{off}, and ρ;
$R_e^1, R_e^2, \ldots, R_e^k$, a set of workload spikes that are sorted in the descending order of their values;
G, the maximum number of groups
Output: $\mathcal{S}$, an ordered partition

1: $\mathcal{S} \leftarrow \{k\}$
2: $r_{min} \leftarrow MinN[k] \times R_e^1$
3: Let $x_0 \leftarrow 0$ for convenience
4: **for** $g = 2$ to G **do**
5: Generate all g-partitions of k (using Eq. (11))
6: **for** each g-partition $\{x_1, \ldots, x_g\}$ **do**
7: **for** each permutation $x'_1, \ldots, x'_g$ of $x_1, \ldots, x_g$ **do**
8: $r \leftarrow \sum_{i=1}^{g} MinN[x'_i] \times \max\{R_e^j | \sum_{h=0}^{i-1} x'_h < j \leq \sum_{h=0}^{i} x'_h\}$
9: **if** $r < r_{min}$ **then**
10: $r_{min} \leftarrow r$, $\mathcal{S} \leftarrow \{x'_1, \ldots, x'_g\}$
11: **end if**
12: **end for**
13: **end for**
14: **end for**
15: **return** $\mathcal{S}$;

We use $\mathcal{S}$ to record the best partition so far (line 1), and use r_{min} to record the amount of resources needed by that partition $\mathcal{S}$ (line 2). For each integer g ($2 \leq g \leq G$), we first generate all possible g-partitions of k using Equ. (11); then, for each permutation $x'_1, ..., x'_g$ of a g-partition $x_1, ..., x_g$, we compute the amount of resources needed by this ordered partition (line 8) and compare it with r_{min}: if this ordered partition uses fewer resources than $\mathcal{S}$, we update $\mathcal{S}$ and r_{min} (lines 9-11). Finally, the optimal ordered partition $\mathcal{S}$ is returned.

It takes $O(p(k))$ time to generate all possible partitions of an integer k [34]. Since $p_g(k) \sim \frac{k^{g-1}}{g!(g-1)!}$, generating all possible g-partitions ($1 \leq g \leq G$) requires $O(\sum_{g=1}^{G} \frac{k^{g-1}}{g!(g-1)!})$ time. Taking $G = 3$ for example, since $p_1(k) = O(1)$, $p_2(k) = O(k)$, and $p_3(k) = O(k^2)$, generating all possible g-partitions ($1 \leq g \leq 3$) requires $O(n^2)$ time. For each possible permutation of a partition, evaluating the amount of resources needed requires $O(k)$ time, thus, the total time complexity of

Ordered partition	The amount of resources used
$\{4\}$	$MinN[4] \times 13 = 26$
$\{1,3\}$	$MinN[1] \times 13 + MinN[3] \times 10 = 33$
$\{3,1\}$	$MinN[3] \times 13 + MinN[1] \times 9 = 35$
$\{2,2\}$	$MinN[2] \times 13 + MinN[2] \times 10 = 23$
$\{1,1,2\}$	$MinN[1] \times 13 + MinN[1] \times 10 + MinN[2] \times 10 = 33$
$\{1,2,1\}$	$MinN[1] \times 13 + MinN[2] \times 10 + MinN[1] \times 9 = 32$
$\{2,1,1\}$	$MinN[2] \times 13 + MinN[1] \times 10 + MinN[2] \times 9 = 32$

Fig. 9. All possible ordered partitions on PM H_2 in Fig. 8, where the optimal ordered partition result is $\{2,2\}$.

Algorithm 3 is $O(k\sum_{g=1}^{G}\frac{k^{g-1}}{(g-1)!})$. In practice, we can choose a proper G to achieve a balance between complexity and optimality.

We use PM H_2 in Fig. 8 as an example, where the sizes of the four spikes are 13, 10, 10, and 9. Without loss of generality, we let $R_e^1 = 13$, $R_e^2 = 10$, $R_e^3 = 10$, and $R_e^4 = 9$. We consider all ordered one-partition, two-partitions, and three-partitions of $k = 4$, i.e., $G = 3$. Fig. 9 shows the results, where the optimal ordered partition is $\{2,2\}$.

7 Performance Evaluation

In this section, we conduct extensive simulations and testbed experiments to evaluate the proposed algorithms under different settings and reveal insights of the proposed design performance.

7.1 Simulation Setup

Two commonly-used packing strategies are considered here, which both use the First Fit Decrease heuristic for VM placement. The first strategy is to provision VMs for peak workload (FFD by R_p), while the second is to provision VMs for normal workload (FFD by R_b). Provisioning for peak workload is usually applied for the initial VM placement [1], where cloud tenants choose the peak workload as the fixed capacity of the VM to guarantee application performance. On the other hand, provisioning for normal workload is usually applied in the consolidation process, since at runtime the majority of VMs are in the OFF state, i.e., most of the VMs only have normal workloads.

We consider both the situations without and with live migration, where different metrics are used to evaluate the runtime performance. For experiments without live migration, where only local resizing is allowed to dynamically provision resources, we use the capacity overflow ratio defined in Section 4 as the performance metric. Next, in our testbed experiments, we add live migration to our system to simulate a more realistic computing cluster, in which the number of migrations reflects the quality of performance, and the number of active PMs reflects the level of energy consumption.

7.2 Simulation Results

We first evaluate the computation cost of our algorithm briefly, and then quantify the reduction of the number of running PMs, as well as compare the runtime performance with two commonly-used packing strategies.

To investigate the performance of our algorithm in various settings, three kinds of workload patterns are used for each experiment: $R_b = R_e$, $R_b > R_e$ and $R_b < R_e$, which denote workloads with normal spike size, small spike size, and large spike size, respectively. It will be observed later that, the workload pattern of VMs does affect the packing result, number of active PMs, and number of migrations.

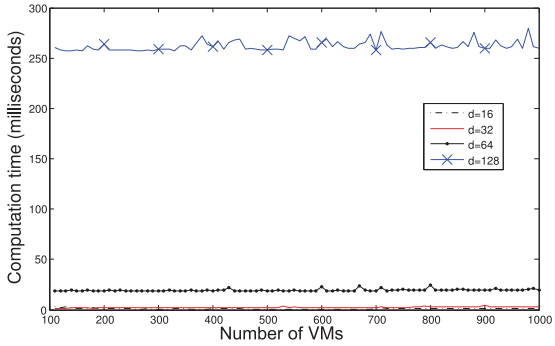


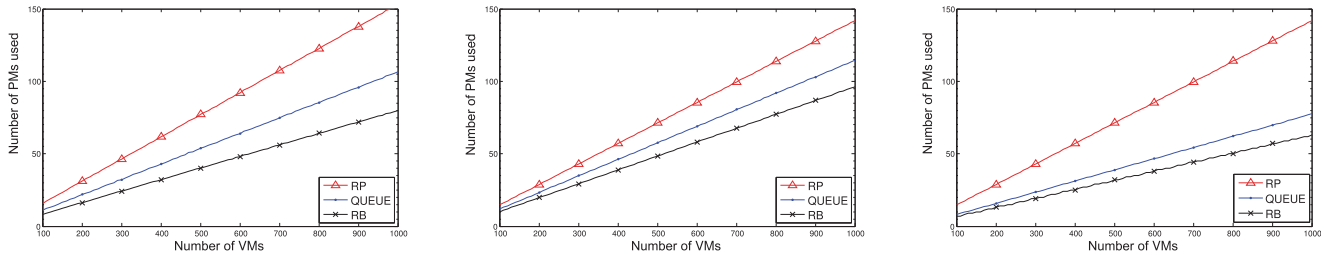
Fig. 10. The computation cost of QUEUE with varying d and n. The cost of the actual placement varies with different physical configurations and thus is not included.

According to the results in Section 5.3, the time complexity of QUEUE is $O(d^4 + n\ log\ n + mn)$. In Fig. 10, we present the experimental computation cost of QUEUE with reasonable d and n values. We see that, our algorithm incurs very few overheads with moderate n and d values. The cost variation with respect to n is not even distinguishable in the millisecond-level.

To evaluate the consolidation performance of QUEUE in different settings, we then choose R_b and R_e uniformly and randomly from a certain range for each VM. We repeat the experiments multiple times for convergence. The capacity overflow ratio is used here as the metric of runtime performance. Since FFD by R_p never incurs capacity violations, it is not included in the performance assessment.

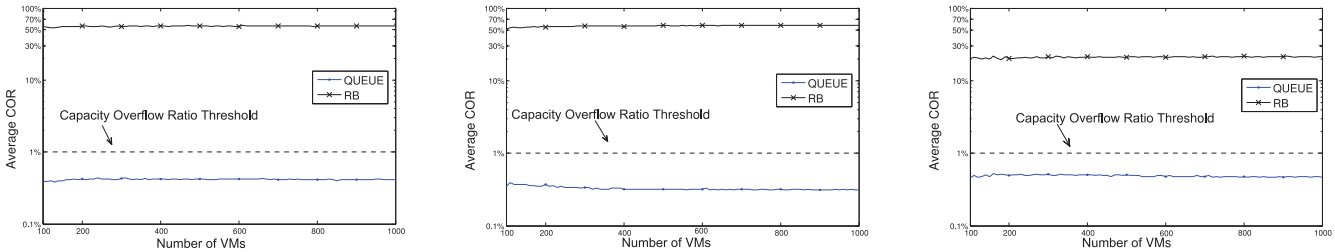
Figs. 11 and 12 show the packing results and COR results, respectively. The common settings for three sub-figures are as follows: $\rho = 0.01$, $d = 16$, $p_{on} = 0.01$, $p_{off} = 0.09$, and $C_j \in [80, 100]$. As we discussed in Section 3, p_{on} indicates the frequency of spike occurrence. For a bursty workload, the spikes usually occur with low frequency and short duration, therefore, we choose $p_{on} = 0.01$ and $p_{off} = 0.09$. Workload patterns are distinguished via setting different ranges for R_b and R_e. For Figs. 11a and 12a, $R_b = R_e$, R_b and $R_e \in [2, 20]$; for Figs. 11b and 12b, $R_b > R_e$, $R_b \in [12, 20]$, $R_e \in [2, 10]$, and for Figs. 11c and 12c, $R_b < R_e$, $R_b \in [2, 10]$, $R_e \in [12, 20]$.

We see that QUEUE significantly reduces the number of PMs used, as compared with FFD by R_p (denoted as RP). When $R_b < R_e$, the number of PMs used in QUEUE is reduced by 45 percent compared with RP, where the ratios for $R_b = R_e$ and $R_b > R_e$ are 30 and 18 percent, respectively. FFD by R_b (denoted as RB) uses even fewer PMs, but the runtime performance is disastrous according to Fig. 12. The COR of RB is unacceptably high. With larger spike sizes ($R_b < R_e$), the packing result of QUEUE is better, because more PMs are saved compared with RP, and fewer additional PMs (for live migrations) are used compared with RB (see Fig. 11c). Simultaneously, with larger spike sizes, the average COR of QUEUE is slightly higher, but is still bounded by ρ (see Fig. 12c). The case of smaller spike sizes shows the opposite results.



(a) $R_b = R_e$ (b) $R_b > R_e$ (c) $R_b < R_e$

Fig. 11. Packing results. The common settings are: $\rho = 0.01$, $d = 16$, $p_{on} = 0.01$, $p_{off} = 0.09$, and $C_j \in [80, 100]$. (a) $R_b = R_e$, R_b and $R_e \in [2, 20]$. (b) $R_b > R_e$, $R_b \in [12, 20]$, $R_e \in [2, 10]$. (c) $R_b < R_e$, $R_b \in [2, 10]$, $R_e \in [12, 20]$.



(a) $R_b = R_e$ (b) $R_b > R_e$ (c) $R_b < R_e$

Fig. 12. Comparison results of QUEUE and RB with respect to capacity overflow ratio.

We mention that, there are very few PMs with CORs slightly higher than ρ in each experiment. This is because a Markov chain needs some time to enter into its stationary distribution. Though we did not theoretically evaluate whether the chain constructed in Section 5 is rapid-mixing, in our experiments, we find that the time period before the chain enters into its stationary distribution is very short.

7.3 Testbed Experiment

We use Xen Cloud Platform (XCP) 1.3 [35] as our testbed to enable live migration in our system. XCP is an open-source cloud platform of its commercial counterpart XenServer. Our proposed scheme can be easily integrated into any existing enterprise-level computing cloud since it simply computes the amount of reserved resources on each PM. A total of 15 machines (Intel Core i5 Processor with four 2.8 GHz cores and 4 GB memory) are used. Ubuntu 12.04 LTS Server Edition is installed both on the PMs and VMs. The resource type in QUEUE can be any one-dimensional resource such as CPU, memory, disk I/O, network bandwidth, or any combination of them that can be mapped to one dimension. For simplicity, memory is designated as the resource type concerned in out testbed experiments. Dynamic scheduling is integrated into our testbed, thus to automatically scale up/down on-demand, as well as to conduct live migration when local resizing is not capable of allocating enough resources.

Fig. 13 shows the architecture of our testbed. We developed three main modules: consolidation module, performance monitor, and schedule module. QUEUE is implemented in the consolidation module. To construct the $MinN$ array, QUEUE gets VM specifications from cloud users and the predetermined system parameters (e.g., ρ, and d) from XCP API. The $MinN$ array can be reused as long as all of k, p_{on}, p_{off}, and ρ remain unchanged. The FidOptPat algorithm (Algorithm 3) may be time-consuming. To improve time-efficiency of QUEUE, we can only invoke FidOptPat when a PM does not have enough physical resources to accommodate another VM. The second module is responsible for periodically retrieving performance statistics from XCP API and forwarding them to the schedule module, which makes decisions about local resizing and live migration, and sends these decisions to XCP API.

We also develop programs in VMs to simulate web servers dealing with computation-intensive user requests. When the number of users visiting the server is more than usual, a workload spike occurs. Users send their requests to the server from time to time, and the time interval between two consecutive requests from the same user follows negative exponential distribution with the mean being 1. Since in reality this interval cannot be infinitely small, we set a lower limit of 0.1. The workload is quantified by the number of requests and each VM generates its workload with its respective R_b and R_e. Fig. 14 shows a sample of the synthetic workload.

We are also interested in studying the effect of different workload patterns, thus, R_b and R_e are classified into three types: *small* (S), *medium* (M), and *large* (L). A certain amount of users can be accommodated for each size—400 for *small*, 800 for *medium* and 1,600 for *large*. Fig. 15 shows the details of various workload patterns in our testbed experiments.

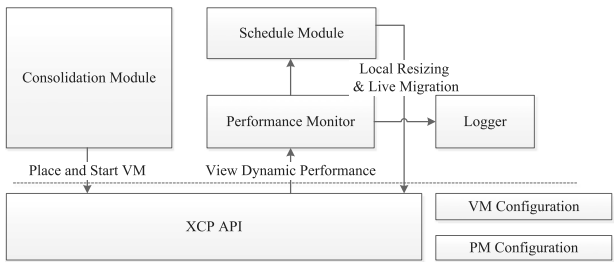


Fig. 13. The architecture of our testbed.

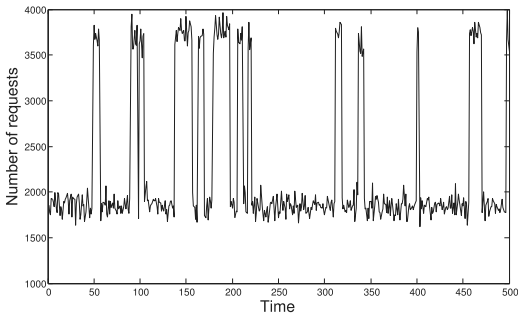


Fig. 14. Sample of the synthetic workload used in our testbed experiment.

7.4 Testbed Experiment Results

The packing results in the testbed experiments are consistent with those in Fig. 11, so we choose not to repeat them. Allowing live migration in our testbed makes the number of PMs used vary over time, so we record the number of PMs used and the number of migrations in the evaluation period. Generally speaking, if a web server runs for a very long time, it probably will not quit in the future, so we assume that the number of PMs used and the number of migrations remain unchanged after the evaluation period. Hence more PMs used at the end of the evaluation period mean more overall energy consumption. Therefore, we use them—the total number of migrations and the number of active PMs at the end of the evaluation period—as the performance metrics. We update the measurements of these metrics every $\sigma = 30$ seconds, and the length of the evaluation period is 100σ seconds. In fact, we observe that, the system becomes stable within about 10σ seconds.

For each workload pattern, we compare the runtime performance of three consolidation strategies—QUEUE, RB, and a simple burstiness-aware algorithm (denoted as RB-EX). RB-EX is simply to reserve at least δ percent of all resources on each PM, which is an applicable consolidation strategy when, in reality, nothing about the workload pattern (except the existence of burstiness) is known. In our experiments, we choose $\delta = 30\%$. The result is averaged over 10 executions for convergence. Fig. 16 shows the comparison results. At the end of the evaluation period, on average, RB uses fewer PMs than QUEUE, but it incurs many more migrations than QUEUE; the performance of RB-EX is between RB and QUEUE. These performance gaps attenuate in $R_b > R_e$ and enlarge in $R_b < R_e$.

We also investigate the time-order patterns of migration events. As shown in Fig. 17, in general, QUEUE incurs very few migrations throughout the evaluation period. At the beginning of the evaluation period, RB and RB-EX incurs excessive migrations due to the over-tight initial VM placement, and the number of PMs used increases rapidly during this period. RB incurs an unacceptably large number of

Pattern	R_b	R_e	Number of users	
			Normal	Peak
$R_b > R_e$	M	S	800	1200
	L	M	1600	2400
$R_b = R_e$	S	S	400	800
	M	M	800	1600
	L	L	1600	3200
$R_b < R_e$	S	M	400	1200
	M	L	800	2400

Fig. 15. Various workload patterns in our experiments.

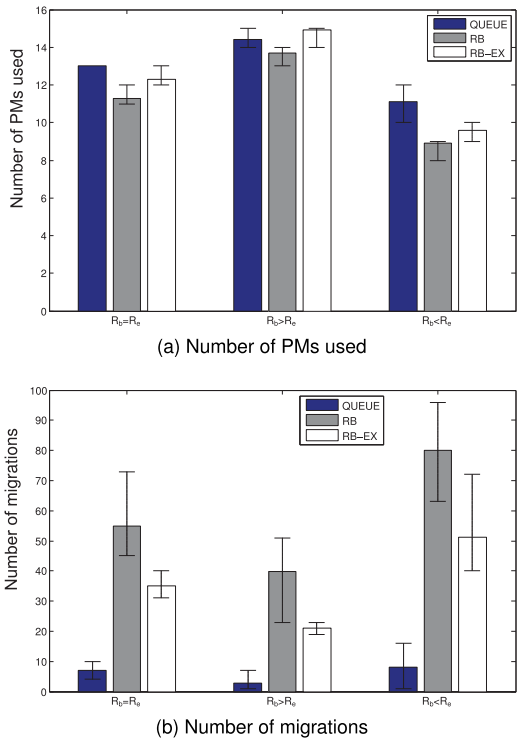


Fig. 16. Bars show the average values, and the extended whiskers show the maximum and minimum values. ($\rho = 0.01$, $p_{on} = 0.01$, $p_{off} = 0.09$, $\sigma = 30s$, $\delta = 0.3$ for RB-EX, the length of evaluation period is 100σ, and VM configurations are set based on Fig. 15.)

migrations throughout the evaluation period, while RB-EX either incurs considerable number of migrations constantly, and uses only slightly more PMs than RB, or incurs very few migrations as QUEUE and uses more PMs than QUEUE (sometimes uses the same number of PMs as QUEUE).

To explain this phenomenon, we introduce a term *idle deception* to refer to the situation where a PM is falsely reckoned idle. In a highly-consolidated cloud, idle deception is very likely to happen, i.e., a busy PM is likely to be selected as a migration target. As a result, the over-provisioned PM tends to become the source PM of migration later, which causes a vicious feedback circle where migrations occur constantly inside the system, while the number of PMs used keeps at a low level. We call this phenomenon *cycle migration*. The results of RB-EX are more subtle. As we have observed, two kinds of results are possible for RB-EX depending on different experiment settings: (1) RB-EX uses

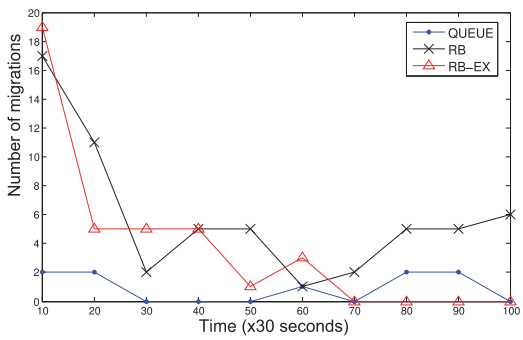


Fig. 17. Comparison of time-order patterns of migration events during one of the experiments for $R_b = R_e$. Similar results are observed for $R_b > R_e$ and $R_b < R_e$.

slightly more PMs than RB, while cycle migration still exists like in RB; and (2) in RB-EX, cycle migration disappears, but more PMs are used than QUEUE. From this point of view, RB-EX performs less efficiently than QUEUE.

7.5 Summary

Key observations are summarized as follows.

1) QUEUE reduces the number of active PMs by up to 45 percent with large spike size ($R_b < R_e$) and up to 30 percent with normal spike size ($R_b = R_e$) in comparison with provisioning for peak workload.
2) QUEUE incurs very few migrations, while both RB and RB-EX incur excessive migrations at the beginning of each experiment due to the over-tight initial packing, and the number of PMs used in RB or RB-EX increases rapidly during this period.
3) Due to falsely picking migration targets, i.e., idle deception, RB incurs an unacceptably large number of migrations constantly throughout the experiment, and the overall performance is seriously degraded.
4) RB-EX performs less efficiently than QUEUE, while either cycle migration exists or cycle migration disappears, but more PMs are used than QUEUE.

8 Discussions

Overhead of learning parameters. One limitation of the two-state Markov chain model is that, learning parameters requires computing clouds to provide tentative deployments, which may incur additional overhead to clouds. However, this overhead can be drastically reduced if tenants have to reserve resources for the same type of VMs repeatedly and lastingly. For example, about 40 percent of applications are recurring in Bing's production data centers [36]. For the same type of VMs, the cloud provider only needs to offer one tentative deployment, and the same results could be fed back to tenants who want to deploy that type of VM. Thus, the tentative deployment overhead for cloud providers would be greatly reduced.

Capacity overflow due to inaccurate workload model. This happens when the model training phase is too short to represent the actual workload pattern of a VM. This kind of capacity overflow is not the main problem we consider in the paper; instead, we assume that it is the cloud user's responsible to adjust its model parameters to best fit its objective. When the cloud user finds the requested resource is not enough to support his/her application, the user may enlarge its model parameters; otherwise, when the user finds that a part of the requested resource is idle most of the time, the user may reduce its model parameters. In this paper, we are interested in bounding the ratio of the capacity overflow due to insufficient queueing blocks, and we use queueing theory to derive the minimal number of queueing blocks that ensure probabilistic performance guarantee on each PM.

Different p_{on}'s and p_{off}'s. In QUEUE, we assume that all VMs have the same state switch probabilities, because the problem becomes very challenging when VMs have various p_{on}'s and p_{off}'s. It has been proved in [22] that the bin packing problem becomes #P-complete even when the sizes of items follow the Bernoulli distribution—a simplified version of two-state Markov chain. Therefore, it is extremely hard to have approximate solutions for situations with different p_{on}'s and p_{off}'s. In practice, we can cluster VMs based on their p_{on}'s and p_{off}'s (i.e., p^i_{on} and p^i_{off} are the 2-D coordinates of V_i in a plane) [37], and apply QUEUE to each cluster.

Online problem. We emphasize that QUEUE can easily adapt to the online situation. When a new VM arrives, we place it on the first PM that satisfies the constraint in Eq. (10), and recalculate the size of the queueing system; When a VM leaves, we simply recalculate the size of the queueing system on the PM; When a batch of new VMs arrives, we use the same scheme as Algorithm 2 to place them. Additionally, if p_{on} and p_{off} varies among VMs, we need to round them to uniform values. In this situation, VM arrival and VM leave may affect the accuracy of the rounded p_{on} and p_{off} values, which requires periodical recalculation of p_{on} and p_{off}.

Multi-dimensional resource. The resource type in the proposed algorithm is one-dimensional; here, we outline how to transform it into a multi-di mensional version. If each dimension of resources is correlated, we can map them to one dimension and apply the proposed algorithm without any major modifications. Otherwise, Algorithm 1 could be applied to each dimension and could be used to quantify the amount of reserved resources for each dimension independently. In the latter case, QUEUE in Algorithm 2 is not applicable, so we need to use another heuristic such as First Fit to place VMs on PMs. The multi-dimension issue with bursty workload is left as part of future work.

9 Conclusion

In a highly consolidated computing cloud, the VM performance is prone to degradation without an appropriate VM placement strategy, if various and distinct burstiness exists. To alleviate this problem, we have to activate more PMs, leading to more energy consumption. To balance the performance and energy consumption with respect to bursty workload, we propose to reserve a certain amount of resources on each PM that form a queueing sytem to accommodate burstiness. To quantify the amount of reserved resources is not a trivial problem. In this paper, we propose a burstiness-aware server consolidation algorithm based on the two-state Markov chain. We use a probabilistic performance constraint and show that the proposed algorithm is able to guarantee this performance constraint. The simulation and testbed results show that, QUEUE improves the consolidation ratio by up to 45 percent with large spike size and around 30 percent with normal spike size, as compared to those provisioning for peak workload, and a better balance of performance and energy consumption is achieved in comparison with other commonly-used schemes.

Acknowledgments

The authors thank the reviewers for their insightful suggestions. This work was supported in part by NSFC (61472181, 61202113, 61321491, and 91218302), EU FP7 IRSES MobileCloud Project(612212), Key Project of Jiangsu Research Program (BE2013116), China Postdoctor Science Fund (2015M570434) and Collaborative Innovation Center of Novel Software Technology and Industrialization. Zhuzhong Qian is the corresponding author.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] M.-H. Oh, S.-W. Kim, D.-W. Kim, and S.-W. Kim, "Method and architecture for virtual desktop service," U.S. Patent 20 130 007 737, 2013.

[3] M. Marzolla, O. Babaoglu, and F. Panzieri, "Server consolidation in clouds through gossiping," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw.*, pp. 1–6.

[4] W. Vogels, "Beyond server consolidation," *ACM Queue*, vol. 6, no. 1, pp. 20–26, 2008.

[5] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag.*, 2007, pp. 119–128.

[6] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. ACM 9th ACM SIGCOMM Conf. Internet Meas. Conf.*, 2009, pp. 202–208.

[7] N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "Injecting realistic burstiness to a traditional client-server benchmark," in *Proc. IEEE 6th Int. Conf. Auton. Comput.*, 2009, pp. 149–158.

[8] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *Proc. ACM SIGCOMM*, 2012, pp. 199–210.

[9] S. Zhang, Z. Qian, J. Wu, S. Lu, and L. Epstein, "Virtual network embedding with opportunistic resource sharing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 816–827, Mar. 2014.

[10] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *Proc. IEEE INFOCOM*, 2011, pp. 71–75.

[11] A. Verma, G. Kumar, and R. Koller, "The cost of reconfiguration in a cloud," in *Proc. 11th Int. Middleware Conf. Ind. Track*, 2010, pp. 11–16.

[12] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proc. IEEE CloudCom*, 2009, pp. 254–265.

[13] Z. Luo and Z. Qian, "Burstiness-aware server consolidation via queuing theory approach in a computing cloud," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, 2013, pp. 332–341.

[14] S. M. Ross, *Introduction to Probability Models*, 10th ed. Orlando, FL, USA: Academic, 2011.

[15] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[16] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement," in *Proc. IEEE Int. Conf. Services Comput.*, 2011, pp. 72–79.

[17] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "HARMONY: Dynamic heterogeneity-aware resource provisioning in the cloud," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 510–519.

[18] S. Zhang, Z. Qian, J. Wu, and S. Lu, "SEA: Stable resource allocation in geographically distributed clouds," in *Proc. IEEE Int. Conf. Commun.*, 2014, pp. 2932–2937.

[19] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. IEEE INFOCOM 2012*, pp. 963–971.

[20] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proc. ACM CoNEXT*, 2010, pp. 15:1–15:12.

[21] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proc. IEEE 31st Int. Conf. Distrib. Comput. Syst.*, 2011, pp. 559–570.

[22] J. Kleinberg, Y. Rabani, and É. Tardos, "Allocating bandwidth for bursty connections," *SIAM J. Comput.*, vol. 30, no. 1, pp. 191–217, 2000.

[23] A. Goel and P. Indyk, "Stochastic load balancing and related problems," in *Proc. IEEE 40th Annu. Symp. Found. Comput. Sci.*, 1999, pp. 579–586.

[24] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective VM sizing in virtualized data centers," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. 2011*, pp. 594–601.

[25] D. Breitgand and A. Epstein, "Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds," in *Proc. IEEE INFOCOM*, 2012, pp. 2861–2865.

[26] Z. Gong, X. Gu, and J. Wilkes, "PRESS: Predictive elastic resource scaling for cloud systems," in *Proc. Int. Conf. Netw. Service Manag.*, 2010, pp. 9–16.

[27] R. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and QoS in cloud computing environments," in *Proc. IEEE Int. Conf. Parallel Process.*, 2011, pp. 295–304.

[28] G. Casale, N. Mi, and E. Smirni, "Model-driven system capacity planning under workload burstiness," *IEEE Trans. Comput.*, vol. 59, no. 1, pp. 66–80, Jan. 2010.

[29] G. Casale, N. Mi, L. Cherkasova, and E. Smirni, "Dealing with burstiness in multi-tier applications: Models and their parameterization," *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1040–1053, Sep./Oct. 2012.

[30] Z. Huang and D. Tsang, "SLA guaranteed virtual machine consolidation for computing clouds," in *Proc. IEEE Int. Conf. Commun.*, 2012, pp. 1314–1319.

[31] V. V. Vijay, *Approximation Algorithms*. New York, NY, USA: Springer, 2003.

[32] G. H. Hardy, E. M. Wright, D. R. Heath-Brown, and J. H. Silverman, *An Introduction to the Theory of Numbers*, vol. 4. Oxford, U.K.: Clarendon, 1979.

[33] G. E. Andrews, *The Theory of Partitions*. Cambridge, U.K.: Cambridge Univ. Press, 1976.

[34] I. Stojmenović and A. Zoghbi, "Fast algorithms for generating integer partitions," *Int. J. Comput. Math.*, vol. 70, no. 2, pp. 319–332, 1998.

[35] Xen cloud platform [Online]. Available: http://www.xenproject.org/, Sep. 2012.

[36] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou, "Re-optimizing data-parallel computing," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, pp. 281–294.

[37] R. Xu and D. Wunsch I., "Survey of clustering algorithms," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 645–678, May 2005.

Sheng Zhang received the BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. Currently, He is an assistant professor in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. His research interests include cloud computing and mobile networks. To date, he has published more than 15 papers, including those appeared in the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, ACM MobiHoc, and IEEE INFOCOM. He received the Best Paper Runner-Up Award from IEEE MASS 2012. He is a member of the IEEE.

Zhuzhong Qian received the PhD degree in computer science in 2007. He is an associate professor in the Department of Computer Science and Technology, Nanjing University, P.R. China. His current research interests include cloud computing, distributed systems, and pervasive computing. He is the chief member of several national research projects on cloud computing and pervasive computing. He has published more than 30 research papers in related fields. He is a member of the IEEE.

Zhaoyi Luo received the BS degree from Nanjing University in 2013. He is a Master student (MMath) in the David Cheriton School of Computer Science, University of Waterloo, Canada. His research interests include computer-aided tools and techniques for analyzing software requirements and specifications.

Jie Wu (F'09) is the chair and a Laura H. Carnell professor in the Department of Computer and Information Sciences, Temple University. He is also an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University. Prior to joining Temple University, he was a program director at the National Science Foundation and was a Distinguished Professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including the *IEEE Transactions on Service Computing* and the *Journal of Parallel and Distributed Computing*. He was the general co-chair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as a program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished visitor, ACM distinguished speaker, and the chair for the IEEE Technical Committee on Distributed Processing (TCDP). He received the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He is a CCF distinguished speaker and a fellow of the IEEE.

Sanglu Lu received the BS, MS, and PhD degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published more than 80 papers in referred journals and conferences in the above areas. She is a member of the IEEE.