2013 IEEE 27th International Symposium on Parallel & Distributed Processing

Burstiness-aware Server Consolidation via Queuing Theory Approach in a Computing Cloud

Zhaoyi Luo
State Key Laboratory for Novel Software Technology
Software Institute
Nanjing University, P.R.China
Email: luozy09@software.nju.edu.cn

Zhuzhong Qian
State Key Laboratory for Novel Software Technology
Department of Computer Science and Technology
Nanjing University, P.R.China
Email: qzz@nju.edu.cn

***Abstract*—Burstiness is a common pattern of virtual machines (VMs)'s workload in production data centers, where spikes usually occur aperiodically with low frequency and last shortly. Since virtualization technology enables elastic resource provisioning in a computing cloud, the bursty workloads could be handled effectively through dynamically scaling up/down. However, to cut back energy consumption, VMs are usually highly consolidated with the minimum number of physical machines (PMs) used. In this case, to meet the runtime expanding demands of the resources (spikes), some VMs have to be migrated to other idle PMs, which is costly and causes performance degradation potentially. In this paper, we investigate the elastic resource provisioning problem and propose a novel VM consolidation mechanism with resource reservation which takes burstiness into consideration as well as energy consumption. We model the resource requirement pattern as the popular ON-OFF Markov chain to represent burstiness, based on which a reservation strategy via queuing theory approach is given for each PM. Next we present a complete VM consolidation scheme with resource reservation within reasonable time complexity. The experiment result show that our algorithms improve the consolidation ratio by up to 45% with large spike size and around 30% with normal spike size compared to those provisioning for peak workload, and a better balance of performance and energy consumption is achieved in comparison with other commonly used consolidation algorithms.**

***Keywords*-Virtual machine placement; Server consolidation; Bursty workload; Stochastic Process; Queuing theory**

I. Introduction

Virtualization is a crucial technique in modern data centers, which enables one server 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. In a computing cloud, the cost of energy consumption (for illumination, power supply, cooling, etc.) occupies a significant fraction of the total operating costs [1]. Therefore, making optimal utilization of underlying resources to reduce the energy consumption is becoming an important issue [2], [3]. To cut back energy consumption, server consolidation aims at packing VMs tightly to reduce the number of PMs used, but performance may be seriously affected if VMs are not appropriately placed especially in a highly consolidated cloud.

It has been shown in previous work [2], [4], [5] that variability and burstiness of VM workload widely exists in modern data centers. For instance, burstiness may be caused in a web server by flash crowed with bursty incoming requests. VMs should be provisioned with resources commensurate with their workload requirements [6], which is a more complex issue considering workload variation. As is shown in Figure 1 two kinds of provisioning strategies are commonly used with workload burstiness – provisioning for peak workload and provisioning for normal workload. Provisioning for peak workload is a favor to guarantee VM performance, but it undermines the benefits of elasticity and leads to low resource utilization [7]. In contrast, provisioning for normal workload takes advantage of elasticity in cloud computing, but it is prone to cause performance degradation in a highly consolidated cloud where resource contention is generally prominent among VMs. In this case, reserving extra resources on each PM is an effective way to improve performance.

To meet the dynamic resource requirements of VMs, local resizing and live migration are pervasively used methods. Local resizing adaptively adjust VM configuration according to the real-time requirement with neglectable time and resource overheads [8]. On the other hand, live migration moves a VM to a relatively idle PM with near-zero downtime when local resizing is not capable to allocate enough resources. However, in a nearly oversubscribed system significant downtime is observed for live migration which also incurs noticeable CPU usage on the host PM [9], which further degrades the performance of collocated VMs. If resources are reserved on the PM to accommodate bursty workload, when a spike occurs the VM can be reconfigured quickly to the new level of resource requirement via local resizing with minimal overheads, instead of being migrated to other PMs. Thus with resource reservation the number of migrations can be reduced considerably leading to performance improvement.

1530-2075/13 $26.00 © 2013 Crown Copyright
DOI 10.1109/IPDPS.2013.62

In this paper, we address the problem of reserving the minimal amount of resources on each PM during consolidation to accommodate bursty workloads while the overall performance is guaranteed. We denote the $capacity\ violation\ ratio$ (CVR) as the fraction of time that the aggregated loads of a PM exceed its capacity. It is assumed that if CVR exceeds a threshold ρ (capacity overflow), one of the hosted VMs needs to be migrated to an idle PM via live migration. Imposing such a threshold ρ rather than conducting migration upon PM's capacity overflow is also a way to tolerate minor fluctuation of resource usage (like the case of CPU usage). We introduce the $performance\ constraint$ as: CVRs of all the PMs should be bounded by ρ (thus to expect no live migration to occur). We use a two-state Markov chain to represent the burstiness of workloads [5], and develop a novel algorithm based on queuing theory techniques – resources reserved on the PM are abstracted as blocks (serving windows in queuing theory) to accommodate spikes. Our goal is to reduce the number and size of blocks on each PM while satisfying the performance constraint.

The contributions of our paper are as follows.

- We propose the idea to reserve resources on each PM to accommodate bursty workload and give a formulated problem description based on the two-state Markov chain model.

- We give an algorithm to quantify the amount of reserved resources based on queuing theory techniques and present a complete VM consolidation scheme.

- Extensive experiments are conducted to validate the effectiveness of our proposed algorithms in comparison with other commonly used consolidation schemes.

The remainder of this paper is organized as follows. Section II is the related work. Section III presents the workload model and problem formulation. Section IV describes our algorithms and complexity. Section V shows the experimental result and Section VI concludes our paper.

II. Related Work

In most of the research work VM consolidation [1], [10], [11] is regarded as a bin-packing problem to map VMs onto PMs with minimal number of PM used. Comprehensive studies on VM packing problems are conducted in [12] and [13]. In [14] and [15], a two-step mapping strategy is used – step one, cluster VMs and map the clusters onto PM aggregates, and step two, map each VM onto a specific PM, and we use a similar idea in our placement scheme. Several works [6], [10], [16]–[18] use stochastic bin-packing (SBP) techniques to deal with variation of workload, where workload is modeled as random variable. In contrast, our two-state Markov chain model takes the additional dimension of time into consideration, and it describes the characteristics of spikes more precisely as we will discuss in Section III.

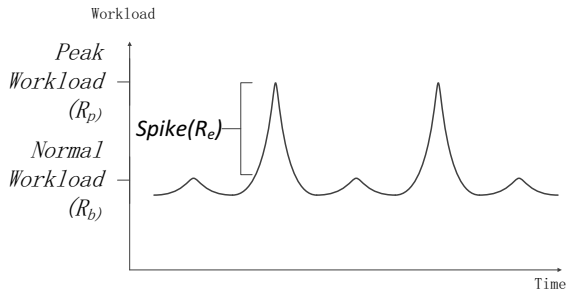


Figure 1. Sample workload trace with burstiness characteristic. Two levels of resource provisioning are presented – normal workload and peak workload.

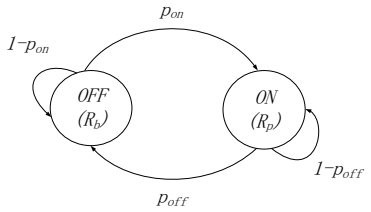


Figure 2. Two-state Markov chain. State ON represents traffic surge (R_p) while state OFF represents the normal level of traffic (R_b). p_{on} and p_{off} are the state switch probabilities.

Recently, [6], [10], [18] study the SBP problem assuming the workload follows normal distribution. In comparison, in our model a lower limit of provisioning is set at the normal workload level which effectively prevents VM interference caused by unpredictable malicious behavior from collocated VMs. In the consolidation framework by Huang and Tsang [19], a constant level of hardware resource is reserved on the PM to tolerate workload fluctuation, but how much resource should be reserved is not given. To the best of our knowledge, no research work has tried to quantify the amount of reserved resources with consideration of distinct workload burstiness.

In a computing cloud, burstiness of workload widely exists in real applications which becomes an inevitable characteristic in server consolidation [2], [4], [5], [7], [20]. Mi et al. [5] models workload as two-state Markov chain to inject burstiness into a traditional benchmark. Several works [3], [21], [22] study modeling and dynamic provisioning of bursty workload in cloud computing.

III. Modeling and Problem Formulation

In order to represent burstiness characteristics, we model VM workload as a two-state Markov chain, where the probability distribution of future states depends upon the current state (see Figure 2 for illustration). In the rest of the paper, we denote R_p, R_b and R_e as the amount of resource requirement by peak workload, normal workload, and the size of spike respectively, where R_e=R_p-R_b as demonstrated in Figure 1. In this model, state ON represents traffic surge

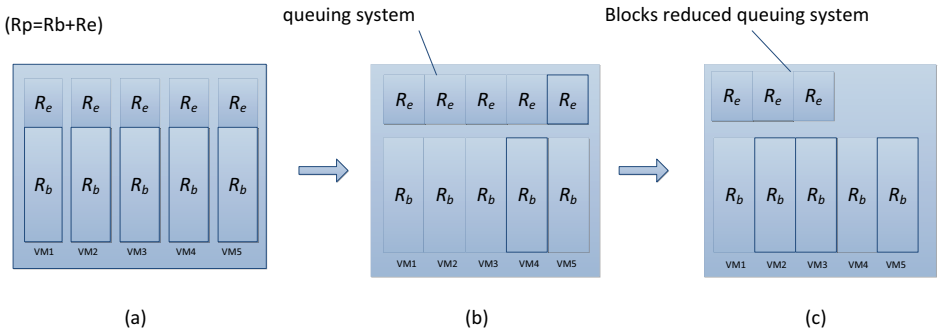


Figure 3. Evolution of the queuing system. (a)The original provisioning strategy for peak workload. (b)gathering all the R_es together to form a queuing system. (c)reducing the number of blocks in the queuing system.

where the amount of resource requirement is R_p, and state OFF represents the normal level of traffic where the amount of resource requirement is R_b. We use p_{on} and p_{off} to denote the state switch probabilities. More specifically, if a VM is in state ON, then the probability it switches to OFF at the next time is p_{off}, and remains ON is $1 - p_{off}$. Similarly if a VM is in state OFF, then the probability it switches to ON at next time is p_{on} and remains 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, p_{on} denotes the frequency of spike occurrence and p_{off} denotes the duration of a spike.

Thus each VM can be described by a four-tuple

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

and each PM can be described by its capacity

$$H_j = (C_j), 1 \leq j \leq m \tag{2}$$

where m and n are the number of PMs and VMs respectively.

Hence a VM placement can be described by a binary mapping $\boldsymbol{X} = [x_{ij}]_{n\times m}$, where $x_{ij} = 1$ 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 amount of resource requirement of V_i at time t. We impose H_j's *capacity constraint* as: the aggregated resource requirement of hosted VMs is no greater than the capacity of H_j.

$$\sum_{i=1}^{n} x_{ij} W_i(t) \leq C_j \tag{3}$$

This is the constraint that should be satisfied for the initial VM placement where $t = 0$. The capacity constraint is said to be violated at time t when Equation (3) is not satisfied. Formally, we denote $vio(j, t) = 1$ if H_j's capacity constraint is violated at time t, and 0 otherwise. Next we define our measurement of performance *capacity violation ratio* (CVR) of H_j as the fraction of time that H_j is violated,

$$CVR_j = \frac{\sum_t vio(j,t)}{t} \tag{4}$$

Obviously smaller CVR_j value implies better performance of H_j.

Our goal is to guarantee the overall performance during consolidation. This is formulated as bounding CVRs of all PMs by a threshold ρ.

$$CVR_j \leq \rho, \ \forall j \tag{5}$$

In summation, the consolidation problem can be formulated as follows:

Given m, n, $\{V_i\}$ and $\{H_j\}$, find a VM-to-PM mapping $\boldsymbol{X}$ to minimize the number of PMs used while satisfying the capacity constraint in Equation (3) for $t = 0$ and the performance constraint in Equation (5) for all t.

$$\begin{aligned} minimize \quad & |\{j | 1 \leq j \leq m, \sum_{i=1}^{n} x_{ij} > 0\}| \quad s.t. \\ & Equation\ (3),\ t = 0 \\ & Equation\ (5),\ \forall t \end{aligned} \tag{6}$$

IV. Algorithms and Complexity

A. Intuition

We propose a novel algorithm to solve the consolidation problem based on queuing theory techniques. Reserving certain amount of resources on each PM to accommodate spikes is the key to bound CVR, and our intuition is to abstract the reserved spaces as blocks (serving windows in queuing theory). We give an informal illustration of the evolution process of our queuing system in Figure 3. Initially VMs are packed by R_p, and each VM has its own block (denoted as R_e in Figure 3). Normally a VM uses only its R_b part, but when a spike occurs, the extra R_e is put in use. Then note that collocated R_es altogether form a queuing system – when a spike occurs the VM enters one of the blocks(R_es) for accommodation, and when the spike disappears it leaves the block until another spike occurs in the future. We should notice that no waiting space exists in the queuing system, so the PM capacity constraint would be violated if a spike occurs while all the blocks are busy, which never happens when the number of blocks equals the number of hosted

VMs. However, we may find a certain number of blocks are idle for the majority of the time, so we can cut off some blocks only incurring very few violations. Therefore, our goal comes to reserving minimal number of blocks on the PM while the performance constraint in Equation (5) is still satisfied.

B. Resource reservation for a single PM

In this part, We focus on resource reservation for a single PM. In order to satisfy the performance constraint in Equation (5), we set the size of a single block uniformly to the maximum of all R_e^is of collocated VMs. Thus the size of reserved resources on a single PM can be determined if hosted VMs and correspondingly the number of blocks needed are known. For simplicity, we impose the constraint that $p_{on}^i = p_{on}$ and $p_{off}^i = p_{off}$ uniformly for all i (p_{on}, $p_{off} > 0$), and the resource we concerned is one-dimensional. The relaxation of these constraints is discussed at the end of this section.

Suppose there are k VMs on the PM and each VM occupies R_b^i resources initially. We initialize the number of blocks on the PM as k, and our objective is to reduce the number of blocks to K, $K < k$ while the performance constraint is still satisfied. Let $\theta(t)$ be the number of busy blocks at time t, which suggests there are $\theta(t)$ VMs in state ON and $k - \theta(t)$ VMs in state OFF. Let $O(t)$ and $I(t)$ denote the number of VMs that switch state from ON to OFF (leaving the queuing system) and from OFF to ON (entering the queuing system) at time t, respectively. Since the workload of each VM changes independently, $O(t)$ and $I(t)$ are mutually independent and both follows the binomial distribution:

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

$$\Longleftrightarrow$$

$$\begin{cases} Pr\{O(t) = x\} = \binom{x}{\theta(t)} p_{off}^x (1 - p_{off})^{\theta(t)-x} \\ \\ Pr\{I(t) = x\} = \binom{x}{k-\theta(t)} p_{on}^x (1 - p_{on})^{k-\theta(t)-x} \end{cases} \tag{7}$$

Assume that state switch of VMs happens at the end of a time interval. Then we have the recursive relation of $\theta(t)$,

$$\theta(t+1) = \theta(t) - O(t) + I(t) \tag{8}$$

Note that $\{\theta(t), t = 0, 1, 2, ...\}$ is a stochastic process from which we can construct a Markov chain with $k + 1$ states (see Figure 4 for illustration). The stochastic process is said to be in state i when the number of busy blocks is i at time t.

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}. If we define $\binom{x}{y} = 0$ when $x > y$ or $x < 0$, then p_{ij} can be derived as follows.

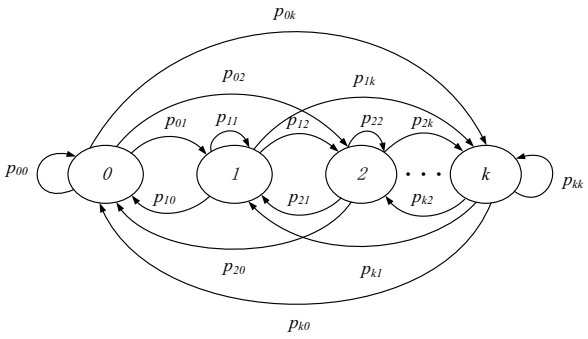


Figure 4. Illustration of constructed Markov chain from $\theta(t)$ with $k + 1$ states .

$$p_{ij} = Pr\{\theta(t+1) = j | \theta(t) = i\} \tag{9}$$

$$= \sum_{r=0}^{i} Pr\{O(t) = r,\ I(t) = j - i + r | \theta(t) = i\} \tag{10}$$

$$= \sum_{r=0}^{i} Pr\{O(t) = r | \theta(t) = i\} \times Pr\{I(t) = j - i + r | \theta(t) = i\} \tag{11}$$

$$= \sum_{r=0}^{i} \binom{r}{i} p_{off}^r (1 - p_{off})^{i-r} \times \binom{j-i+r}{k-i} p_{on}^{j-i+r} (1 - p_{on})^{k-j-r} \tag{12}$$

From the definition of p_{ij} we have Equation (9). Then we get Equation (10) from the recursive relation of $\theta(t)$ in Equation (8). Using the fact that $O(t)$ and $I(t)$ are mutually independent we have Equation (11), and from the distribution of $O(t)$ and $I(t)$ in Equation (7) we have Equation (12).

Let $\boldsymbol{P} = [p_{ij}]$ denote the matrix of one-step transition probabilities which can be calculated by Equation (12) with given p_{on} and p_{off} values. Let k-tuple $\boldsymbol{\Pi}_0 = (1, 0, 0, ..., 0)$ denote the initial state, which indicates that at time $t = 0$, $Pr[\theta(0) = 0] = 100\%$. Then we introduce $\boldsymbol{\Pi} = (\pi_0, \pi_1, \pi_2, ..., \pi_k)$ and let

$$\boldsymbol{\Pi} = \lim_{t \to \infty} \boldsymbol{\Pi}_0 \boldsymbol{P}^t \tag{13}$$

Then π_i is the limiting probability that the stochastic process is in state i, and $\boldsymbol{\Pi}$ is the probability distribution of $\theta(t)$ when t reaches infinity. $\boldsymbol{P}^t$ is the matrix of t-step transition probabilities.

Specifically, π_i also equates the long-run proportion of time that stochastic process is in state i [23]. In our case it means π_i equates the proportion of time at which the number of busy blocks is i.

Next we prove the existence of $\boldsymbol{\Pi}$:

Proposition 1: $\boldsymbol{\Pi}$ defined as above does exist in the context that the Markov chain is constructed from $\theta(t)$.

Proof: Since k is a finite number, the Markov chain constructed from $\theta(t)$ is irreducible. From Equation (13) we know all $p_{ij} > 0$, so all the states are aperiodic. For any state i, if starting in state i it is able to reenter state i within finite time since $p_{ij} > 0$, so all the states are recurrent. In a finite Markov chain all the recurrent states are positive recurrent, and for an irreducible, positive recurrent and aperiodic Markov chain, $\lim_{t\to\infty} \boldsymbol{P}^t$ exists [23]. Hence $\boldsymbol{\Pi}$ exists. ■

Here we use an equivalent equation to get $\boldsymbol{\Pi}$ instead of solving Equation (13).

$$\boldsymbol{\Pi P} = \boldsymbol{\Pi}$$

$$\Longleftrightarrow$$

$$\begin{cases} \sum_{l=0}^{k} \pi_l p_{l0} - \pi_0 = 0 \\ \sum_{l=1}^{k} \pi_l p_{l1} - \pi_1 = 0 \\ ..... \\ \sum_{l=k}^{k} \pi_l p_{lk} - \pi_k = 0 \end{cases} \tag{14}$$

Equation (14) is a typical homogeneous system of linear equations that can be solved by Gaussian elimination.

Finally, we can derive the number of blocks needed from $\boldsymbol{\Pi}$. Let K be the minimum number that $K < k$ and satisfies

$$\sum_{m=0}^{K} \pi_m \geq 1 - \rho \tag{15}$$

If the number of blocks on H_j is reduced from k to K, then the queuing system being in state i, $i \geq k$ at time t indicates $vio(j,t) = 1$, and vice versa. Thus we have

$$\begin{aligned} CVR_j = \lim_{t\to\infty} \frac{\sum_t vio(j,t)}{t} &= \sum_{m=K+1}^{k} \pi_m \\ &= 1 - \sum_{m=0}^{K} \pi_m \leq \rho \end{aligned} \tag{16}$$

Therefore, K is the minimum number of blocks needed on a single PM after reduction to satisfy performance constraint. This algorithm is described in Algorithm 1.

In queuing theory, our model is formalized as a discrete time, finite source, k-window queuing system with geometric service time and no waiting space (finite-source Geom/Geom/k). Relevant queuing theory basics can be found in [23] and detailed analysis on this model is presented in [24].

Algorithm 1 MapCal

Require: input:
The number of VMs on a specific PM, k;
The probability that VM state switches from ON to OFF, p_{off};
The probability that VM state switches from OFF to ON, p_{on};
Ensure: The number of blocks needed, K;
1: Calculate $\boldsymbol{P}$, the matrix of one-step transition probability according to Equation (12);
2: Get the coefficient matrix of the homogeneous system of linear equations described in Equation (14);
3: Solve the equation system via Gaussian elimination to get $\boldsymbol{\Pi}$;
4: Calculate K from $\boldsymbol{\Pi}$ according to Equation (15);
5: **return** K;

Algorithm 2 QueuingFFD

Require: input:
The number of VMs, n;
The number of PMs, m;
The maximum VMs allowed on a single PM, d;
VM specifications, $\{V_i\}$;
PM specifications, $\{H_i\}$;
Ensure: A VM-to-PM placement mapping, $\boldsymbol{X}$;
1: Initialize array $mapping$ of size $d + 1$. Let $mapping[0] \leftarrow 0$;
2: Get p_{on} and p_{off} from V_i;
3: **for** each $k \in [1, d]$ **do**
4: Invoke Algorithm 1 with arguments k, p_{off}, p_{on}, and get the return value K;
5: Let $mapping[k] \leftarrow K$;
6: **end for**
7: Cluster VMs so that VMs with similar R_e are in the same cluster;
8: Sort the clusters according to R_e in descending order;
9: Sort VMs in each cluster according to R_b in descending order;
10: **for** each V_i (by sorted order) **do**
11: place V_i on the first PM H_j that satisfies the constraint in Equation (17), and set $x_{ij} \leftarrow 1$;
12: **end for**
13: **return** $\boldsymbol{X}$;

C. VM consolidation with resource reservation

In this part, we present the complete VM consolidation scheme of n VMs onto m PMs. We tend to locate VMs with similar R_e onto the same PM to reduce the averaged size of a single block, so VMs are consolidated in a two-step manner – first VMs are clustered and sorted, and then each VM is mapped onto a specific PM. We introduce $mapping(k)$, which denotes if there are k VMs on a

PM, then $mapping(k)$ blocks are needed to satisfy the performance constraint on the PM. $mapping(k)$ can be calculated by Algorithm 1 with given k, p_{on} and p_{off}. We assume that a single PM can host up to d VMs, so $mapping(k)$ can be calculated for all possible k from 1 to d preliminarily, followed by a two-step consolidation scheme analogous to the First Fit Decrease (FFD) heuristics with resource reservation based on $mapping(k)$. It is reminded that the size of reserved resources is block size multiplying block number, where block size is conservatively set to the maximum R_e of the hosted VMs. Thus the constraint to judge whether V_i can be placed on H_j is

$$max\{R_e^i, max\{R_e^s | s \in T_j\}\} \times mapping(|T_j| + 1) + R_b^i + \sum_{s \in T_j} R_b^s \leq C_j \quad (17)$$

T_j denotes the set of indices of VMs that have been already placed on H_j, and C_j is the capacity of H_j. Equation (17) indicates that a VM can be placed on H_j if and only if the sum of the new size of the queuing system and the new total size of R_bs does not exceed the capacity of H_j.

The consolidation scheme is presented in Algorithm 2. As is shown, $mapping(k)$ is calculated in Line 1-6 for each $k \in [1, d]$. In Line 7-8 VMs are clustered so that VMs with similar R_e are in the same cluster, and clusters are sorted in descending order according to R_e. This is a cluster-level heuristics to let collocated VMs have similar R_e thus to minimize the averaged size of blocks on all PMs. Next, in Line 9-12 VMs within each cluster are sorted in descending order according to R_b, and each VM is mapped onto a PM by sorted order with First Fit heuristics. At last the VM-to-PM mapping $\boldsymbol{X}$ is returned.

D. Algorithm Complexity

For Algorithm 1, calculating the matrix of one-step transition probability costs roughly $O(k^3)$ if we assume all constants have been calculated preliminarily. Gaussian elimination costs $O(k^3)$, and calculating K costs $O(k)$. Thus the time complexity of Algorithm 1 is $O(k^3)$. For Algorithm 2, to calculate $mapping(k)$ Algorithm 1 is invoked for each $k \in [1, d]$, so this process costs roughly $O(d^4)$. The following FFD heuristics costs $O(nlogn)$ for the process of sorting and $O(mn)$ for the process of placement. Specific clustering techniques is out of the scope of this paper which is not discussed here, and we use a simple $O(n)$ clustering method in our algorithm. Therefore, the time complexity of the complete consolidation algorithm is $O(d^4+nlogn+mn)$.

E. Discussion

We emphasize that our algorithm can easily adapt to the online situation: when a new VM arrives, we place it on the first PM that satisfies the constraint in Equation (17), and recalculate the size of the queue; when a VM quits, we simply recalculate the size of the queue 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 exit may affect the accuracy of the rounded p_{on} and p_{off} values, which requires periodical recalculation of the rounded p_{on} and p_{off}.

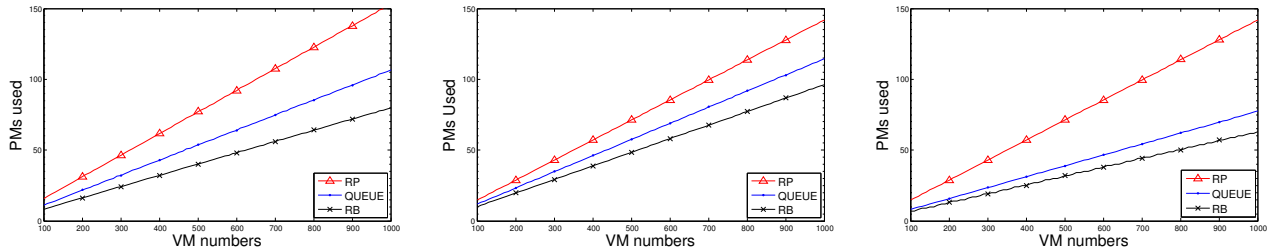
The resource type in the original algorithm is one-dimensional, and here we outline how to transform it into a multi-dimensional version: if each dimension of resources is correlated we can map them to one dimension and apply the original algorithms; otherwise our queuing algorithm should be applied to each dimension and to quantify the amount of reserved resources for each dimension of resource type independently. In this case the original two-step consolidation scheme in Algorithm 2 is not applicable, so we need to use a simpler heuristic such as First Fit and performance constraints should be satisfied on all dimensions. The multi-dimension consolidation issue with bursty workload is included in our future work.

V. Experiment

A. Overview

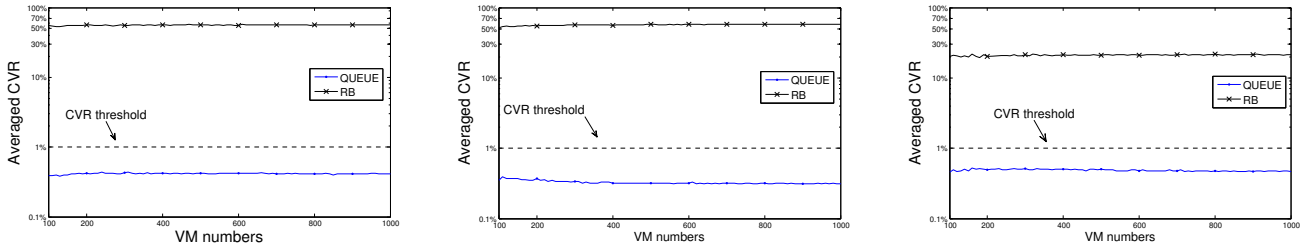
In this section, we first evaluate the computation cost of our algorithm briefly, and then quantify the reduction of the number of used PMs, as well as compare the runtime performance with commonly used packing strategies. Two commonly used packing strategies are considered here which both use the First Fit Decrease heuristic for placement, but the first strategy is to provision VMs for peak workload (FFD by R_p) while the second provision VMs for normal workload (FFD by R_b). Provisioning for peak workload is usually applied for the initial VM placement [7] where cloud users choose the peak workload as the fixed capacity of the VM to guarantee 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 state OFF.

We use Xen Cloud Platform (XCP) 1.3 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 an existing enterprise-level computing cloud since it simply compute an amount of reserved resources on each PM. Both the controller node (the master) and the compute nodes (the slaves) are machines with Intel Core i5 Processor with four 2.8GHz cores and 4 GB memory. Ubuntu 12.04 LTS Server Edition is installed both on the PMs and VMs. The resource we mentioned in the theoretical part can be any one-dimensional resource type such as CPU, memory, disk I/O, network bandwidth, or any combination of them that is mapped to one dimension. For simplicity, memory is designated as the resource type concerned in this section.



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

Figure 5. Packing result. Experiment settings are as follows: $\rho = 0.01$, $d = 16$, $p_{on} = 0.01$, $p_{off} = 0.09$. R_b^i, R_e^i, and PM capacity C_j is randomly generated from a certain range. $C_j \in [80, 100]$. For (a)$R_b = R_e$, $R_b, R_e \in [2, 20]$. For (b)$R_b > R_e$, $R_b \in [12, 20]$ and $R_b \in [2, 10]$. For (c)$R_b < R_e$, $R_b \in [2, 10]$ and $R_e \in [12, 20]$.



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

Figure 6. Performance result for each placement. The performance measurement *capacity violation ratio* (CVR) is the fraction of time that the capacity constraint is violated.

We consider both the situations without and with live migration, in which different measurements are used to evaluate the runtime performance. For experiments without live migration, only local resizing is allowed to dynamically provision resources, and CVR defined in Section III is adopted as the performance measurement. Next we add live migration to our system to simulate a more realistic computing cluster, in which the number of migrations reflects the level of performance and the number of PM used reflects the level of energy consumption.

For variety's sake, three workload patterns are distinguished for each experiment: $R_b = R_e$, $R_b > R_e$ and $R_b < R_e$, which denotes workload with normal spike size, small spike size and large spike size respectively. It is observed that VM's workload pattern do influence the packing result and the performance.

B. Computation cost

As mentioned earlier, the time complexity of our consolidation algorithm is $O(d^4 + nlogn + mn)$. For completeness, we present the experimental computation cost here with reasonable d and n values. The result (see Figure 7) shows 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 millisecond-level.

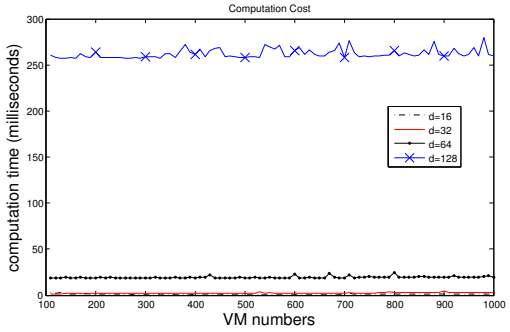


Figure 7. The computation cost of Algorithm 2 with various d and n values to get the placement matrix $\boldsymbol{X}$. The cost of the actual placement process varies with hardware and software configuration and thus is not included.

C. Dynamic scheduling without live migration

In this part we choose R_b and R_e randomly from a certain range for each VM, packing VMs and running them simulatively to assess the performance. Then we repeat the experiment to gather statistical data. The *capacity violation ratio* (CVR) is used here as the measurement of runtime performance. Since FFD by R_p never incurs capacity violations, it is not included in the performance assessment. As we discussed in Section III, p_{on} indicates the frequency of spike occurrence and p_{off} indicates the duration of a spike. For a bursty workload spikes usually occur with low

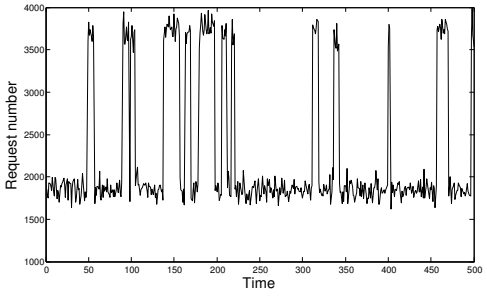


Figure 8. Sample of generated workload used in the experiment

Pattern	R_b	Re	Range of users accommodated	
			normal capability	peak capability
$R_b = R_e$	small	small	400	800
	medium	medium	800	1600
	large	large	1600	3200
$R_b > R_e$	medium	small	800	1200
	large	medium	1600	2400
$R_b < R_e$	small	medium	400	1200
	medium	large	800	2400

Table I
EXPERIMENT SETTINGS ON WORKLOAD PATTERNS

frequency and last shortly, so we choose $p_{on} = 0.01$ and $p_{off} = 0.09$. Workload patterns are distinguished via setting different range of R_b and R_e. (see comments of Figure 5 for detailed experiment settings).

As shown in Figure 5, the packing result of our proposed algorithm (denoted as QUEUE) reduce the number of PMs used significantly compared with FFD by R_p (denoted as RP). For $R_b > R_e$, the number of PMs used is reduced by 45% compared with FFD by R_p, where the ratios for $R_b = R_e$ and $R_b < R_e$ are 30% and 18% respectively. FFD by R_b (denoted as RB) uses even less PMs but the runtime performance is disastrous according to Figure 6 – the CVR is unacceptably high. With larger spike size ($R_b < R_e$), the packing result of QUEUE is better because more PMs are saved compared with RP and less additional PMs are used compared with RB (see Figure 5(c)). Simultaneously, with larger spike size the averaged CVR is slightly higher but still bounded by ρ (see Figure 6(c)). The case of smaller spike size ($R_b > R_e$) shows opposite result. As well we mention the existence of very few PMs with CVRs slightly higher than ρ in each experiment.

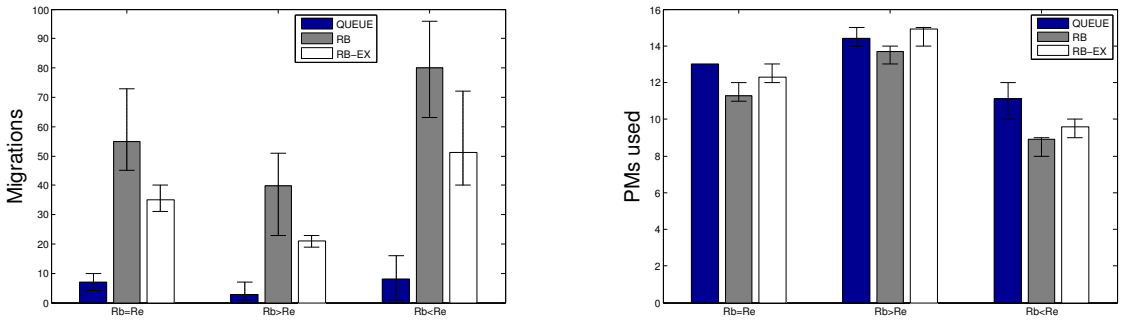
D. Adding live migration to the system

In this part we add live migration to our system to simulate a more realistic computing cluster. Since burstiness may be caused in a web server by flash crowed with bursty incoming requests, we develop programs in VMs to simulate web servers dealing with computation-intensive user requests. When a spike occurs, more users than usual are visiting the server. Users are sending requests to the server periodically, and the period for a user to send request (think time) follows negative exponential distribution with mean=1. Since in reality the user think time cannot be infinitely small, we set a lower limit=0.1. The workload is quantified by request number and each VM generate the workload dynamically with its specific R_b and R_e. Figure 8 shows a sample of the generated workload.

Dynamic scheduling features are integrated into our testbed, thus to automatically scale up/down the resource allocation in a on-demand manner, as well as to conduct live migration when local resizing is not capable to allocate enough resources. To distinguish different workload patterns in this part, R_b and R_e are classified as $small$, $medium$ and $large$, and certain number of customers can be accommodated for each specification – 400 users for $small$, 800 users for $medium$ and 1600 users for $large$. Finally, combinations of R_b and R_e specifications are selected for each workload pattern (see Table I).

The packing result is consistent with that in Figure 5 so it is not presented here. Adding migration to the system, the number of PMs used varies at runtime, so we record the real-time number of PMs used and migration events during the evaluation period. Generally a web server runs for very long time and do not quit usually, so we tend to simulate the process that the system stabilizes after consolidation and assume the number of PMs used and migration events remain stable after the evaluation period. Hence more PMs used at the end of the evaluation period lead to more overall energy consumption during VM life cycle. In the meantime, two measurements are proposed for evaluation – the total number of migrations and the number of PMs used at the end of the evaluation period, which reflect the level of performance and the level of energy consumption respectively. We choose time interval σ=30s as the information update period, and the length of evaluation period is 100σ. Actually, it is observed that the system have stabilized merely within 10σ or so.

Three workload patterns $R_b = R_e$, $R_b > R_e$, $R_b < R_e$ are distinguished, and for each pattern we compare the runtime performance of three packing strategies – packing by our proposed scheme (denoted as QUEUE), FFD by R_b (denoted as RB), and a simple burstiness-aware algorithm (denoted as RB-EX). RB-EX is simply to reserve at least δ-percentile resources on each PM, which is an applicable consolidation strategy in reality when nothing about the workload pattern is known except the existence of burstiness. Specifically we choose $\delta = 0.3(30\%)$. We run each experiment setting for 10 times and gather the statistical result. Average values, as well as the maximum and minimum values of measurements are presented in Figure 9.



(a) migrations (b) pms used

Figure 9. Performance result. ρ=0.01, p_{on}=0.01, p_{off}=0.09, $\sigma = 30s$, evaluation period=100σ, $\delta = 0.3$ for RB-EX, and VM specifications are selected according to Table I. Bars show the average values and extended whiskers show the maximum and minimum values.

As is shown in Figure 9(a), RB incurs unacceptably more migrations than QUEUE, while RB-EX alleviates this problem to some extent. This trend attenuates in $R_b > R_e$ and enlarges in $R_b < R_e$. Figure 9(b) shows at the end of the evaluation period RB commonly uses less PMs than QUEUE. We introduce the term *idle deception* to refer to the situation that a PM is falsely reckoned idle by the system unaware of workload burstiness. In a highly consolidated cloud, idle deception is likely to happen so that a busy PM is selected as the migration target. In turn, this 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 experimental result of RB-EX show more subtle phenomenon. As we have observed, two kinds of result are possible for RB-EX depending on different experiment settings – (i) slightly more PMs than RB are used, while cycle migration still exists like in RB or (ii) cycle migration disappears but more PMs than QUEUE are used. Hence we conclude that RB-EX performs not as well as our consolidation scheme.

Next we investigate in the time-order patterns of migration events. As illustrated in Figure 10, QUEUE incurs very few migrations throughout the evaluation period. At the beginning of an evaluation period RB and RB-EX incurs excessive migrations due to the over-tight initial packing, and the number of PMs used increases rapidly during this period. RB incurs unacceptably large number of migrations throughout the evaluation period, while RB-EX either incurs considerable number of migrations constantly and use only slightly more PMs than RB, or incurs very few migrations as QUEUE but larger (sometimes equal) number of PMs are used than QUEUE.

Finally, we conclude our experimental observations as follows: (i) QUEUE reduce the number of PMs used by 45% with large spike size and 30% with normal spike size

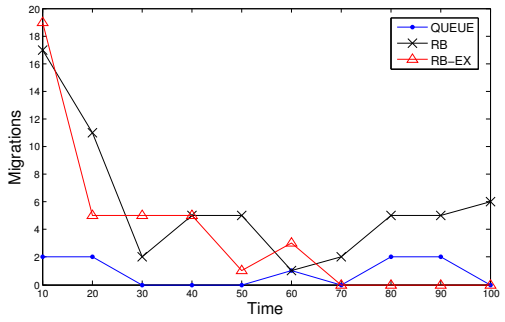


Figure 10. Comparison of time-order patterns of migration events. This sample is selected from one of the experiments for $R_b = R_e$. Similar pattern is also observed for $R_b > R_e$ and $R_b < R_e$.

compared with RP (ii) QUEUE incurs very few migrations throughout the experiment (iii) Both RB and RB-EX incur excessive migrations at the beginning of an experiment due to the over-tight initial packing, and the number of PMs used increases rapidly during this period. (iv) RB incurs unacceptably large number of migrations constantly throughout the experiment, and the overall performance is seriously degraded. (v) As the cause of excessive number of migrations, the phenomenon of cycle migration is observed in RB due to falsely picking migration target. (vi) RB-EX performs not as well as QUEUE, while either cycle migration still exists or cycle migration disappears but more PMs are used than QUEUE. (vii) For larger spike size ($R_b < R_e$), the packing result of QUEUE is better while the performance is slightly worse than those of normal spike size ($R_b = R_e$), whereas $R_b < R_e$ shows opposite result.

VI. Conclusion

Burstiness is a common pattern of VM workload in production data centers, and server consolidation also plays an important role in cloud computing. Both topics have been studied extensively for years, but no work explicitly takes

workload burstiness into consideration during the consolidation process. On the other hand, in a highly consolidated cloud VM performance is prone to degradation without appropriate VM placement if distinct burstiness exists. To alleviate this problem we have to use more PMs than needed which leads to more energy consumption. To balance the performance and energy consumption with respect to bursty workload, certain amount of resources need to be reserved on the PM to accommodate burstiness, and to quantify the amount of reserved resources is not a trivial work. In this paper we propose a burstiness-aware VM consolidation scheme based on the two-state Markov chain. We formulate the performance constraint and show that our proposed algorithm is able to guarantee this performance constraint. The experiment result show that our algorithms improve the consolidation ratio by up to 45% with large spike size and around 30% with normal spike size compared to those provisioning for peak workload, and a better balance of performance and energy consumption is achieved in comparison with other commonly used consolidation schemes.

ACKNOWLEDGMENT

This work is partially supported by the National Natural Science Foundation of China under Grant No. 61073028, 61021062, 61202113; the National Basic Research Program of China (973) under Grant No. 2009CB320705; Jiangsu Natural Science Foundation under Grant No. BK2011510.

REFERENCES

[1] M. Marzolla, O. Babaoglu, and F. Panzieri, "Server consolidation in clouds through gossiping," in *WoWMoM*, 2011, pp. 1–6.

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

[3] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *IM*, 2007, pp. 119–128.

[4] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *ACM SIGCOMM conference on IMC*, 2009, pp. 202–208.

[5] N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "Injecting realistic burstiness to a traditional client-server benchmark," in *ICAC*, 2009, pp. 149–158.

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

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, UC Berkeley, Tech. Rep., 2009.

[8] A. Verma, G. Kumar, and R. Koller, "The cost of reconfiguration in a cloud," in *Middleware Industrial Track*, 2010, pp. 11–16.

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

[10] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective vm sizing in virtualized data centers," in *IM*, 2011, pp. 594–601.

[11] O. Sukwong, A. Sangpetch, and H. Kim, "Sageshift: Managing slas for highly consolidated cloud," in *INFOCOM*, 2012, pp. 208–216.

[12] A. Shankar and U. Bellur, "Virtual Machine Placement in Computing Clouds," 2010.

[13] K. Mills, J. Filliben, and C. Dabrowski, "Comparing vm-placement algorithms for on-demand clouds," in *CloudCom*, 2011, pp. 91–98.

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

[15] 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 *SCC*, 2011, pp. 72–79.

[16] J. Kleinberg, Y. Rabani, and E. Tardos, "Allocating bandwidth for bursty connections," in *Theory of computing*, 1997, pp. 664–673.

[17] A. Goel and P. Indyk, "Stochastic load balancing and related problems," in *Foundations of Computer Science*, 1999, pp. 579–586.

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

[19] Z. Huang and D. Tsang, "SLA Guaranteed Virtual Machine Consolidation for Computing Clouds," in *ICC*, 2012.

[20] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *ICDCS*, 2011, pp. 559–570.

[21] G. Casale, N. Mi, and E. Smirni, "Model-driven system capacity planning under workload burstiness," *Transactions on Computer*, pp. 66–80, 2010.

[22] G. Casale, N. Mi, L. Cherkasova, and E. Smirni, "Dealing with burstiness in multi-tier applications: Models and their parameterization," *Transactions on Software Engineering*, pp. 1040–1053, 2012.

[23] S. M. Ross, *Introduction to Probability Models, Tenth Edition*. Orlando, FL, USA: Academic Press, Inc., 2011, ch. 4, pp. 191–193,195–196,214–216.

[24] N. Tian, X. Xu, and M. Zhanyou, *Discrete Time Queuing Theory (Chinese Edition)*. Beijing, CHINA: Science Press, 2008, ch. 3, pp. 60–65.