CompVM: A Complementary VM Allocation Mechanism for Cloud Systems

Haiying Shen, *Senior Member, IEEE, Member, ACM*, and Liuhua Chen

***Abstract*— In cloud datacenters, effective resource provisioning is needed to maximize the energy efficiency and utilization of cloud resources while guaranteeing the service-level agreement (SLA) for tenants. To address this need, we propose an initial virtual machine (VM) allocation mechanism (called CompVM) that consolidates complementary VMs with spatial/temporal awareness. Complementary VMs are the VMs whose total demand of each resource dimension (in the spatial space) nearly reaches their host's capacity during VM lifetime period (in the temporal space). Based on our observation of the existence of VM resource utilization patterns, the mechanism predicts the resource utilization patterns of VMs. Based on the predicted patterns, it coordinates the requirements of different resources and consolidates complementary VMs in the same physical machine (PM). This mechanism reduces the number of PMs needed to provide VM service, hence increases energy efficiency and resource utilization, and also reduces the number of VM migrations and SLA violations. We further propose a utilization variation-based mechanism, a correlation coefficient-based mechanism, and a VM group-based mechanism to match the complementary VMs in order to enhance the VM consolidation performance. Simulation based on two real traces and real-world testbed experiments shows that CompVM significantly reduces the number of PMs used, SLA violations, and VM migrations of the previous resource provisioning strategies. The results also show the effectiveness of the enhancement mechanisms in improving the performance of the basic CompVM.**

***Index Terms*— Cloud, service-level agreement (SLA), virtual machine (VM), resource provisioning.**

I. Introduction

CLOUD computing has been intensively studied recently due to its great promises [1]. Cloud providers use virtualization technologies to allocate Physical Machine (PM) resources to tenant Virtual Machines (VMs) based on their resource (e.g., CPU, memory and bandwidth) requirements. The scale of modern cloud datacenters has been growing and current cloud datacenters contain tens to hundreds of thousands of computing and storage devices running complex applications. Energy consumption thus become critical concerns. Maximizing energy efficiency and utilization of cloud resources while satisfying Service Level Agreement (SLA) for tenants requires effective management of resource.

Manuscript received May 8, 2016; revised February 5, 2017, May 26, 2017, and December 5, 2017; accepted March 30, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor H. Zheng. Date of publication April 16, 2018; date of current version June 14, 2018. This work was supported in part by the U.S. NSF under Grant OAC-1724845, Grant ACI-1719397, and Grant CNS-1733596 and in part by the Microsoft Research Faculty Fellowship under Grant 8300751. An early version of this work was presented in the Proceedings of Infocom 2014 [50]. *(Corresponding author: Haiying Shen.)*

H. Shen is with the Computer Science Department, University of Virginia, Charlottesville, VA 22904-4740 USA (e-mail: hs6ms@virginia.edu).

L. Chen is with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634 USA.

Digital Object Identifier 10.1109/TNET.2018.2822627

Previous server resource provisioning (or VM allocation) strategies can be classified into two categories: static provisioning and dynamic provisioning [2]. Static provisioning [3]–[7] allocates physical resources to VMs only once based on static VM resource demands, which can be reduced to a bin-packing problem. However, reserving VM peak resource requirement for the entire execution time cannot fully utilize resources. In order to more fully utilize cloud resources, dynamic provisioning [8]–[14] has been proposed, which first consolidates VMs using a certain strategy with a resource requirement lower than the peak and then uses live VM migration to handle PM overload to mitigate SLA violations [8]. VM migration generates overhead and degrades VM performance. Therefore, when a PM is not overloaded, it may not be necessary to conduct VM migration. These VM allocation strategies only consider resource demands at one or each time point. Therefore, they fail to coordinate the resource requirements in different resource dimensions (in the spatial space) for a period of time (in the temporal space); that is, they are spatial/temporal-oblivious, which fails to constantly fully utilize different resources. Some previously proposed resource provisioning methods consider spatial balance (e.g., [15]–[18]) or temporal correlations (e.g., [19], [20]) but do not simultaneously consider both to fully utilize different resources over time.

Our primary goal is to handle the aforementioned problems and design a VM allocation mechanism to further reduce the number of PMs needed for service provisioning, maximize resource utilization and reduce the number of VM migrations, while ensuring SLA guarantees. To this end, we propose an initial VM allocation mechanism (called CompVM) that predicts the VM resource utilization patterns and consolidates complementary VMs with spatial/temporal-awareness. Complementary VMs are the VMs whose total demand of each resource dimension (in the spatial space) nearly reaches their host PM's capacity during VM lifetime period (in the temporal space). For example, a low-CPU-utilization and high-memory-utilization VM and a high-CPU-utilization and low-memory-utilization VM can be consolidated in one PM to fully utilize both of its CPU and memory resources. As shown in a simple 1-dimensional resource space (i.e., one resource type) in Figure 1(a), the resource utilization patterns of VM1, VM2 and VM3 are complementary to each other on the resource. Placing these three VMs together in the PM can fully utilize this resource of the PM and avoid PM overloads while still ensuring the SLA guarantees. In Figure 1(b), by consolidating VM3, VM4 and VM1, the CPU and memory resources of this PM are fully utilized. Currently there are various types of VMs (e.g., CPU-intensive, memory-intensive, data-intensive) and they consume different ratios of resource capacities for different resource types. Therefore, it is reasonable to assume the existence of different VMs that are complementary to each other.

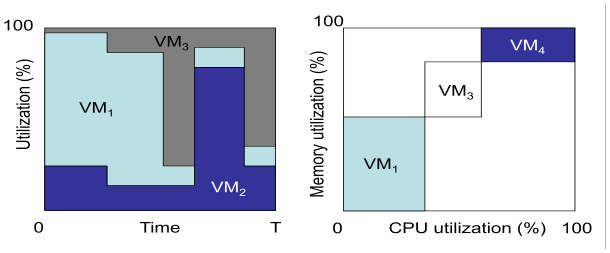


Fig. 1. Consolidating complementary VMs in one PM. (a) In temporal space. (b) In spatial space.

Application and job are interchangeable terms in this paper. A job consists of a number of VMs (or tasks) (VM and task are interchangeable terms in this paper). For example, a MapReduce job consists of several tasks. A Web service application consists of many VMs. Each job in the PlanetLab and Google Cluster VM traces [21], [22] consists of many of tasks. It was indicated that when VMs are configured to run an application collaboratively, their workload pattern variations can be predicted [23]. We notice that different VMs running the same short-term job task (e.g., MapReduce) tend to have similar resource utilization patterns, because each VM executes exactly the same source code with the same options. In long-term applications such as web services and file services, the workloads on the VMs are often driven by human requests determined by daily human activities. Therefore, these VMs exhibit periodical resource utilization patterns. Thus, based on the historical resource utilizations of VMs from a tenant, the lifetime resource utilization patterns for short-term VMs or periodical resource utilization patterns for long-term VMs requested by this tenant to run the same job can be predicted. The contribution of this paper can be summarized as follows.

- We study VMs running short-term MapReduce jobs and observe that the VMs running the same job task tend to have similar resource utilization patterns over time. We also study the PlanetLab and Google Cluster VM traces and find that different VMs running a long-term job exhibit similar periodical resource utilization patterns.
- We then design a practical algorithm to detect the resource utilization patterns from a group of VMs.
- We propose an initial VM allocation mechanism that consolidates complementary VMs based on the predicted VM resource demand patterns. The mechanism coordinates the requirements of different resources of the VMs to realize spatial/temporal-aware VM consolidation.
- We propose a utilization variation based mechanism and a correlation coefficient based mechanism to identify complementary VMs and allocate them to PMs, and also propose an alternative initial VM allocation mechanism that combines complementary VMs considering all resource dimensions into a group and assigns the whole group to a PM. These enhancement mechanisms further improve resource utilization.
- We conduct comprehensive simulation based on two real traces and real-world experiments running a MapReduce job. Experimental results show that CompVM significantly reduces the number of PMs, SLA violations and VM migrations.

Note that our work can be used for an environment with any number of resource types and it does not limit the types of resources. The rest of the article is organized as follows. Section II introduces the VM resource demand pattern detection algorithm. Section III presents the details of CompVM and our enhancement mechanisms. Section IV evaluates our method in trace-driven simulation experiments. Section V evaluates our method in real-world testbed. Section VI briefly describes the related work. Finally, Section VII summarizes the paper with remarks on future work.

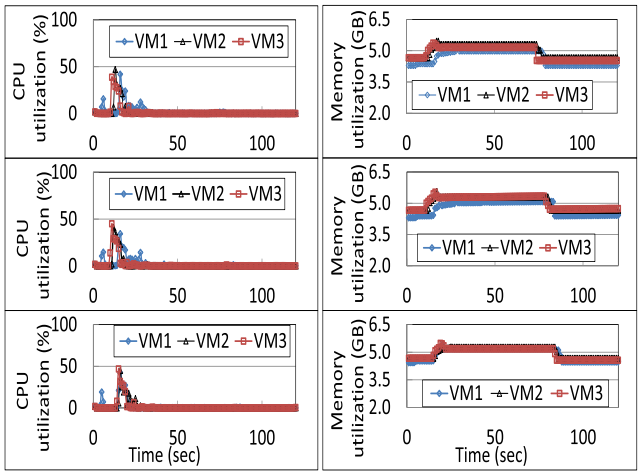


Fig. 2. VM resource utilization for *TeraSort* on three datasets. (a) CPU utilization. (b) Memory utilization.

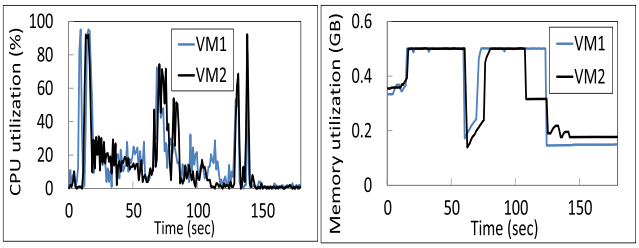
II. VM Resource Utilization Profiling and Pattern Detection

A. Profiling VM Resource Demands

In order to predict the resource demand profiles of cloud VMs, we conducted a measurement study on VM resource utilizations. Workload arrives at the virtual cluster of a tenant in the form of jobs. Usually all tasks in a job execute the same program with the same options. Also, application user activities have daily patterns. Thus, different VMs running the same job tend to have similar resource utilization patterns. To confirm this, we conducted a measurement study on both short-term jobs (from MapReduce benchmarks) and long-term jobs (from the Google Cluster Trace and PlanetLab trace).

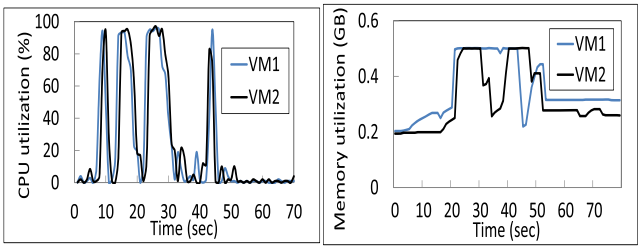
1) Utilization Patterns of VMs for Short-Term Jobs: MapReduce jobs represent an important class of applications in cloud datacenters. We profile the CPU and memory utilization patterns of typical MapReduce jobs. We conducted the profiling experiments on our cluster consisting of 15 machines (3.4GHz Intel(R) i7 CPU, 8GB memory) running Ubuntu 12.04. We constructed a virtual cluster of a tenant with 11 VMs; each VM instance runs Hadoop 1.0.4. We recorded the CPU and memory utilization of each VM every 1 second.

We used *Teragen* to randomly generate 1G data, then ran *TeraSort* to sort the data in the virtual cluster. Figures 2(a) and 2(b) display the resource utilization results of three VMs for different generated datasets. Figure 3 displays the resource utilizations of two VMs running *TestDFSIO write*, which generates 10 output files with each file having 0.1GB. Figure 4 displays the resource utilizations of two VMs running *TestDFSIO read*, that reads 10 input files generated by *TestDFSIO write*. From the figures, we can find that the VMs collaboratively running the same job have



(a) (b)

Fig. 3. VM resource utilization for *TestDFSIO write*. (a) CPU utilization. (b) Memory utilization.



(a) (b)

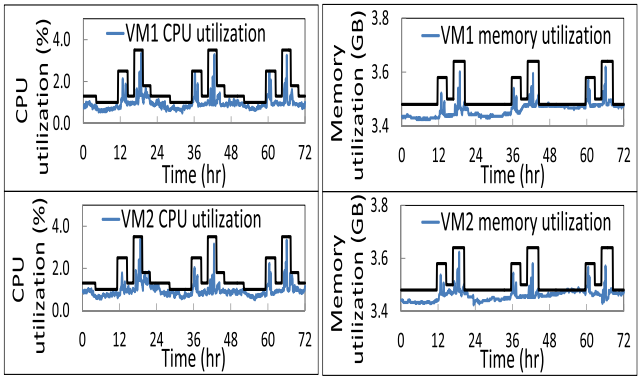
Fig. 4. VM resource utilization for *TestDFSIO read*. (a) CPU utilization. (b) Memory utilization.

similar resource utilization patterns. The VMs running the same job on different datasets also have similar resource utilization patterns. We repeatedly ran each experiment several times and got similar resource utilization patterns for the VMs, which indicates that VMs running the same job task at different times also have similar resource utilization patterns.

2) Utilization Patterns of VMs for Long-Term Jobs: To study the utilization patterns of VMs for long-term jobs, we used publicly available Google Cluster trace [22] and the PlanetLab trace [21]. The Google Cluster trace records resource usage on a cluster of about 11000 machines from May 2011 for 29 days. The PlanetLab trace contains the CPU utilization of a subset of the VMs in PlanetLab every 5 minutes for 24 hours in 10 random days in March and April 2011. We considered a task in the Google Cluster trace as a VM. For the PlanetLab trace, we identified the VMs for the same job by the names of the trace file. For example, trace files with the same file name are VMs that run the same job in different places and times. In the Google Cluster trace, we analyzed 700 VMs and found that different VMs running the same job tend to have similar utilization patterns. Also, for a long-term VM, daily periodical patterns can be observed from the VM trace. We randomly chose two VMs running the same job as an example to show our observations. Figure 5(a) shows the CPU utilizations of two VMs every five minutes during three days and Figure 5(b) shows their memory utilizations. We see that both CPU and memory resource demands exhibit periodicity approximately every 24 hours. Also, the two VMs exhibit similar resource utilization patterns since they collaboratively ran the same job. In the PlanetLab trace, we analyzed 900 VMs and also found that they exhibit daily periodical patterns. Figure 6 shows the CPU utilization of a randomly selected VM to show their periodical patterns.

B. The VM Resource Utilization Pattern Detection Algorithm

The previous section shows the existence of similar resource utilization patterns of VMs running the same job. Given the



(a) (b)

Fig. 5. VM resource utilization from Google Cluster trace. (a) CPU utilization. (b) Memory utilization.

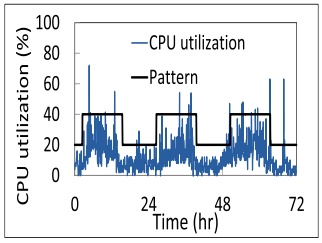


Fig. 6. VM resource utilization from PlanetLab trace.

resource requirement pattern of VMs in an application, we can potentially derive some complicated functions (e.g., high-order polynomials) to precisely model the changing requirement over time. However, such smooth functions significantly complicate the process of VM allocation due to the complexity of model formulation. Also, very accurate pattern modeling of an individual VM cannot represent the general patterns of a group of VMs for similar applications. To achieve a balance between modeling simplicity and modeling precision, we choose to model the resource requirement as simple pulse functions introduced in [24] as shown in Figure 7. These four models sufficiently capture the resource demands of the applications. An actual VM resource demand that is much more complicated usually exhibits a pattern which is a combination of these simple types.

Next, we introduce how to detect the resource utilization pattern for a VM. The cloud records the resource utilizations of the VMs of a tenant. If the job on a VM is a short-term job (e.g., MapReduce job), the cloud records the entire lifetime of the job. If the job on a VM is a long-term job (e.g. Web server VM), the cloud records several periods that show a regular periodical pattern. From the log, the cloud can obtain the resource utilization of VMs of a tenant running the same application. When a tenant issues a VM request to the cloud, based on the resource utilization pattern of previous VMs from this tenant running the same application, the cloud can estimate the resource utilization pattern of this requested VM.

Let $\mathcal{D}_i(t)=(D_i^1(t),..,D_i^d(t))$ $(t = T_0,\ldots,T_0+T)$ be the actual d dimension resource demands of VM i at time t. Given the resource demands of a set of N VMs running the same job from a tenant, $\mathcal{D}_i(t)$ $(i = 1,2,\ldots,N)$, our pattern detection algorithm finds a pattern $\mathcal{P}(t)=(P^1(t),..,P^d(t))$ $(t=T_0,\ldots,T_0+T)$. The derived pattern will be considered as the future resource demand profile of a requested VM from the tenant.

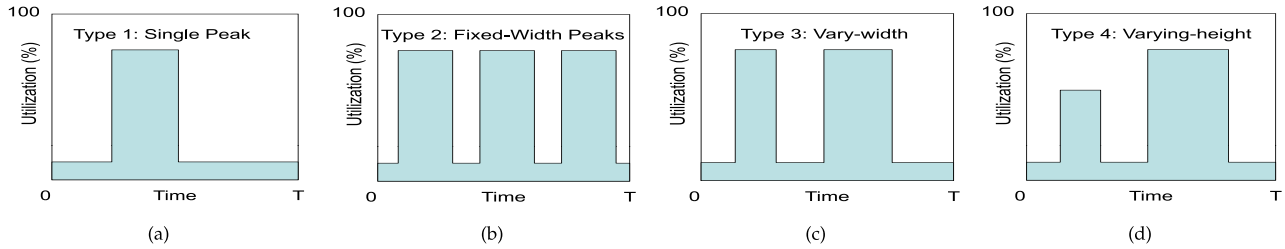


Fig. 7. Time-varying resource utilization classification. (a) Type 1: Single peak. (b) Type 2: Fixed-width peaks. (c) Type 3: Varying-width peaks. (d) Type 4: Varying height peaks.

Algorithm 1 VM Resource Demand Pattern Detection

1: **Input:** $\mathcal{D}_i(t)$: Resource demands of VM $i, i = 1, 2, \ldots, N, t = T_0, \ldots, T_0 + T$
2: **Output:** $\mathcal{P}(t)$: VM resource demand pattern
3: */* Find the maximum demand at each time */*
4: $\mathcal{E}(t) = \max_{i \in \{1,\ldots,N\}} \mathcal{D}_i(t_j)$ for each time t
5: */* Smooth the maximum resource demand series */*
6: $\mathcal{E}(t) \leftarrow$ LowPassFilter($\mathcal{E}(t)$) for each time t
7: */* Use sliding window W to derive pattern */*
8: $\mathcal{P}(t) = \max_{t \in \{t_j, t_j+1, \ldots, t_j+W\}} \mathcal{E}(t_j)$ for each time t
9: */* Round the resource demand values */*
10: $\mathcal{P}(t) \leftarrow$ Round($\mathcal{P}(t_j)$) for each time t
11: **return** $\mathcal{P}(t)$ $(t = T_0, \ldots, T_0 + T)$

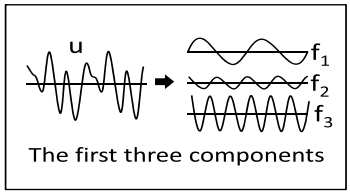


Fig. 8. Fourier decomposition.

Algorithm 1 shows how to generate the resource demand pattern for a requested VM. The algorithm first finds the maximum demand $\mathcal{E}(t)$ among the set of $\mathcal{D}_i(t)$ $(i = 1, 2, \ldots, N)$ at each time t (Line 4). Then, it passes $\mathcal{E}(t)$ through a low pass filter (Line 6) to remove high frequency components to smooth $\mathcal{E}(t)$. In this step, as shown in Figure 8, we use Fast Fourier Transform (FFT) to decompose the pattern to components with different frequencies and then remove the high frequency components. The algorithm then utilizes a sliding window of size W to find the envelop of $\mathcal{E}(t)$ (Line 8). Finally, it rounds the demand values (Line 10). In this step, the demand fraction value is rounded to the demand absolute value.

To evaluate the accuracy of our pattern detection algorithm, we conducted an experiment on predicting VM resource request pattern based on resource utilization records of a group of VMs running the same application from the PlanetLab trace and the Google Cluster trace. We randomly selected 700 jobs and predicted the CPU utilization of a VM in each job during 24 hours. Specifically, in the PlanetLab trace, we used the CPU utilizations of three VMs of a job on March 3rd, 6th and 9th in 2011 to predict the CPU utilization of a VM and compared it with the actual utilization of a VM of the job on March 22nd, 2011. In the Google Cluster trace, we used the CPU and memory utilizations of two VMs of a job on May 1st and 2nd in 2011 to predict the CPU and memory utilizations

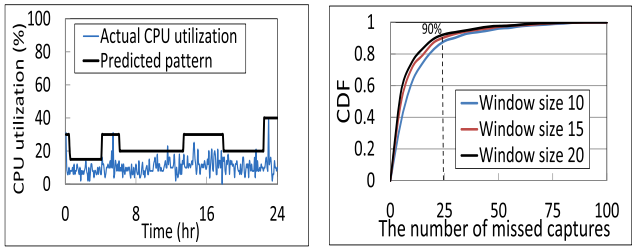


Fig. 9. Pattern detection using the PlanetLab trace. (a) Actual and predicted CPU utilizations. (b) CDF of # of missed captures using the PlanetLab trace.

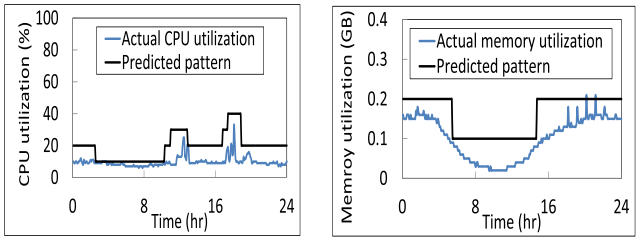


Fig. 10. Pattern detection using the Google Cluster trace. (a) Actual and predicted CPU utilizations. (b) Actual and predicted memory utilizations.

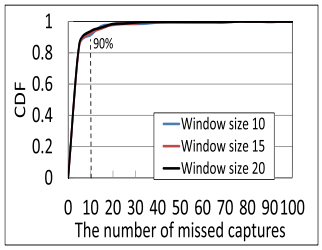


Fig. 11. CDF of # of missed captures using the Google Clusster trace.

of a VM and compared them with the real utilizations of a VM of the job on May 3rd, 2011.

Figure 9(a) displays the actual VM CPU utilization and the predicted pattern generated by our pattern detection algorithm using the PlanetLab trace. Figure 10(a) and Figure 10(b) display the actual VM CPU and memory utilizations and the predicted pattern using the Google Cluster trace. We see that the pattern can capture the utilization most of the time except for a few burst peaks. Most of these burst peaks are only slightly greater than the pattern cap and are single bursts. This means that the resources provisioned according to the pattern can ensure the SLA guarantees most of the time, i.e., before and after the burst points.

When the real VM CPU request from the trace is greater than the predicted value, we say that a *missed capture* occurs. Figure 9(b) and Figure 11 show the cumulated distributed function (CDF) of the number of missed captures

from our 700 predictions using the PlanetLab trace and the Google Cluster trace, respectively. The three curves in the figure correspond to the pattern detection algorithm with different window sizes. We see that up to 90% of the detected patterns have missed captures fewer than 25 during the 24 hours in PlanetLab trace, and up to 90% of the detected patterns have missed captures fewer than 10 in Google Cluster trace. We also see that the patterns generated by a bigger window size generates fewer missed captures compared to a small window size because a larger window size leads to more resource provisioning. As the previous dynamic provisioning strategies, VM migration upon SLA violation is a solution for these missed captures. CompVM helps reduce a large number of VM migrations in the previous dynamic provisioning strategies.

III. Initial VM Allocation Mechanism

CompVM is suitable for an environment that the workloads of the majority VMs can be predicted. CompVM can improve the initial VM allocation of these VMs and hence improve the whole system performance to a certain extent. We assume that the tasks of a job are the same. CompVM can only be used for VMs whose resource utilizations can be predicted and can improve the VM allocation performance for these VMs. Based on previous work [25] and our measurement in the above section, we assume that the VMs of the same job have similar resource utilization patterns. When a new VM is created in the system, CompVM will check if the VMs of this VM's job were created before (based on the job name and/or user ID). If yes, the resource utilization pattern of the new VM can be predicted based on those VMs. If a job has different tasks, CompVM needs to further identify the same tasks in a job. For example, a user can be requested to use an ID to mark the same tasks in a job. If a task's resource utilization is unpredictable, the task is allocated using the original VM allocation algorithm. To consider prediction confidence, we can adopt the method in our previous work in [26]. To predict VM resource utilization without pattern seasonality, we can use our proposed method in [27].

A. Initial VM Allocation Policy Based on Resource Efficiency

The goal of CompVM is to place all VMs in as few hosts as possible, ensuring that the aggregated demand of VMs placed in a host does not exceed its capacity across each resource dimension. We consider the VM consolidation as a classical d-dimensional vector bin-packing problem [28], where the hosts are conceived as bins and the VMs as objects that need to be packed into the bins. This problem is an NP-hard problem [28]. We then use a dimension-aware heuristic algorithm to solve this problem, which takes advantage of cross dimensional complimentary requirements for different resources as illustrated in Figures 1 in Section II-A.

Each host j is characterized by a d-dimensional vector to represent its capacities $\mathcal{H}_j = (H_j^1, H_j^2, \ldots, H_j^d)$. Each dimension represents the host's capacity corresponding to a different resource such as CPU and memory. Recall that $\mathcal{D}_i(t) = (D_i^1(t), D_i^2(t), .., D_i^d(t))$ denotes the actual resource demands of VM i. We define the fractional VM demand vector of VM i on PM j as

$$\begin{aligned}\mathcal{F}_{ij}(t) &= (F_{ij}^1(t), F_{ij}^2(t), \ldots F_{ij}^d(t)) \\ &= (\frac{D_i^1(t)}{H_j^1}, \frac{D_i^2(t)}{H_j^2}, .., \frac{D_i^d(t)}{H_j^d}). \end{aligned} \quad (1)$$

The resource utilization of PM j with N VMs on resource k at time t is calculated by $U_j^k(t) = \frac{1}{H_j^k}\sum_{i=1}^{N} D_i^k(t)$.

In order to measure whether a PM has available resource for a VM in a future period of time, we define the normalized residual resource capacity of a host as $\mathcal{R}_j(t) = (R_j^1(t), R_j^2(t), \ldots, R_j^d(t))$, in which

$$R_j^k(t) = 1 - U_j^k(t) = 1 - \frac{1}{H_j^k}\sum_{i=1}^{N} D_i^k(t). \quad (2)$$

When a VM is allocated to a PM, the VM's fractional VM demand F_{ij}^k and the PM's normalized residual resource capacity R_j^k must satisfy the capacity constraint below at each time t and for each resource k:

$$F_{ij}^k(t) \le R_j^k(t),\ t = T_0', \ldots,\quad T_0' + T,\ k = 1, 2 \ldots, d. \quad (3)$$

in order to guarantee that the host has available resource to host the VM resource request for the time period $[T_0', T_0' + T]$.

For each resource k, we hope that a PM j's $U_j^k(t)$ at each time t is close to 1, that is, its each resource is fully utilized. To jointly measure a PM's resource utilization across different resources at each time, we define the *resource efficiency* during time period $[T_0', T_0' + T]$ as the ratio of the aggregated resource demand over the total resource capacity:

$$E_j^k = \frac{1}{T \cdot H_j^k}\sum_{t=T_0'}^{T_0'+T}\sum_{i=1}^{N} D_i^k(t)dt. \quad (4)$$

We use a norm-based greedy algorithm [29] to capture the distance between the average resource demand vector and the capacity vector of a PM (e.g., the top right corner of the rectangle in the 2-dimensional space):

$$M_j = \sum_{k=1}^{d}\{w_k(1 - E_j^k)\}^2, \quad (5)$$

where w_k is the assigned weight to resource k, which can be determined by resource intensity aware algorithms [30]. For simplicity, we can make all weights the same and set $w_k = 1$. This distance metric coordinately measures the closeness of each resource's utilization to 1.

To identify the PM from a group PMs to allocate a requested VM i, CompVM first identifies the PMs that do not violate the capacity constraint of Equ. (3). It then places the VM i to a PM that minimizes the distance M_j, that is, this VM can more fully utilize each resource in this PM.

Algorithm 2 shows the pseudocode for CompVM. This mechanism refers to the resource demand pattern $\mathcal{P}_i(t)$ from the library that approximately predicts the resource demands of VMs from the same tenant for the same job. Based on $\mathcal{P}_i(t)$ and the host capacity vector $\mathcal{H}_j$, we can derive predicted $\mathcal{F}_{ij}(t)$. For each candidate host (Line 5, where m is the number of host), we first check whether it has enough resource for hosting the VM at each time $t = T_0', \ldots, T_0' + T$ for each resource by comparing $\mathcal{F}_{ij}(t)$ and $\mathcal{R}_j(t)$ (Line 6 and Lines 19-26) in order to ensure that $F_{ij}^k(t) \le R_j^k(t)$ (Equ.(3)) during the VM lifetime or periodical interval $[T_0', T_0' + T]$. If the host has sufficient residual resource capacity to host this VM, then we calculate the resource efficiency (Lines 9-12) after allocating this VM during time period $[T_0', T_0' + T]$ using Equ. (4). Finally, we choose the PM that leads to the minimum distance based on resource efficiency (Lines 13-17).

Algorithm 2 Pseudocode for Initial VM Allocation

```
1: Input: P_i(t): Predicted resource demands
          R_j(t): Residual resource capacity of candidates
2: Output: Allocated host of the VM. It is Null if it cannot be found.
3:     AllocatedHost=Null
4:     M=Double.MAX_VALUE //initialize the distance
5:     for j = 1 to m do
6:         if CheckValid(P(t),R_j(t))==false then
7:             continue
8:         else
9:             for k = 1 to d do
10:                E_j^k = E_j^k + 1/(T·H_j^k) Σ_{t=T'_0}^{T'_0+T} P^k(t)dt
11:                M_j+ = {w_k(1 − E_j^k)}^2
12:            end for
13:            if M_j<M then
14:                M=M_j
15:                AllocatedHost=host j
16:    end for
17:    return AllocatedHost
18:
19: function CheckValid(P(t),R_j(t)):
20:    for k = 1 to d do
21:        for t = T'_0 to T'_0 + T do
22:            if F_ij^k(t) > R_j^k(t) (Equ.(3)==false)
23:            return false
24:        end for
25:    end for
26:    return true
```

It means this VM can make this PM most fully utilize its different resources among the PM candidates. In this way, the complementary VMs are allocated to the same PM, thus fully utilizing its different resources.

B. Enhanced Initial VM Allocation Mechanisms

1) Basic Rationale: In the previous section, the VM allocation mechanism tries to maximize the *resource efficiency* during the monitoring time period based on Equ. (4). However, E_j^k is the average utilization of PM j during the monitoring time period, and it cannot reflect the deviation of the resource utilization during this period. For example, the time period consists of epochs t_1 and t_2. A PM with a resource usage of 10 units at epoch t_1 and a usage of 20 units at epoch t_2 has the same *resource efficiency* as a PM with usages of 15 units at both t_1 and t_2. Let's say we are selecting a PM for hosting a VM from two candidates. The VM demands 10 units of resource at epoch t_1 and 20 units of resource at epoch t_2. The first PM's available capacity is 100 units and 20 units for the two epochs, respectively. The second PM's total capacity is 60 for both epochs. Both candidate PMs have the same *resource efficiency*. If we choose the first PM, the capacity is used up at epoch t_2. It cannot host more VMs though it has available capacity at epoch t_1. Choosing the second PM is preferred as it can still host extra VMs after accepting the VM. In the following, we will introduce three methods to improve the initial VM allocation mechanism.

2) Utilization Variation Based Mechanism: In order to further distinguish PMs, we should measure other metrics instead of only calculating the average E_j^k. We can exam the utilization variation of the estimated utilization curve of a PM j after accepting the VM. We define the variance of a PM j with residual resource $R_j^k(t_i)$ as

$$\sigma^2 = \frac{\sum_{k=1}^{d}\sum_{i=1}^{T}[R_j^k(t_i) - \overline{R_j^k(t_i)}]^2}{T} \tag{6}$$

where $R_j^k(t_i)$ is the residual type-k resource at time t_i, and $\overline{R_j^k(t_i)}$ is the average residual type-k resource. σ^2 is the utilization variation, which measures how far a set of numbers is spread out. We can select PMs that will have identical resource utilization ($\sigma^2 = 0$) between time epochs after accepting the VM based on the utilization variation of the resulting utilization of the PM.

Algorithm 3 Pseudocode for the Utilization Variation Based VM Allocation Mechanism

```
1: Input: P_i(t): Predicted resource demands
          R_j(t): Residual resource capacity of candidates
2: Output: Allocated host of the VM. It is Null if it cannot be found.
3:     AllocatedHost=Null
4:     Var=Double.MAX_VALUE //utilization variation
5:     for j = 1 to m do
6:         if CheckValid(P(t),R_j(t))==false then
7:             continue
8:         else
9:                 Update R_j^k(t_i) and the average R_j^k(t_i)
                   after allocating the VM
10:                σ^2 = Σ_{k=1}^{d} Σ_{i=1}^{T} [R_j^k(t_i) − R_j^k(t_i)]^2 / T
11:            if σ^2<Var then
12:                Var=σ^2
13:                AllocatedHost=host j
14:    end for
15:    return AllocatedHost
```

Algorithm 3 shows the pseudocode for the utilization variation based VM allocation mechanism. Similar to Algorithm 2, this mechanism refers to the resource demand pattern $\mathcal{P}_i(t)$ of VM i and the residual resource capacity $\mathcal{R}_j(t)$ of candidate PM j. For each candidate host, the algorithm first checks whether it has enough resource for hosting the VM for each resource by calling CheckValid($\mathcal{P}(t)$,$\mathcal{R}_j(t)$) (Lines 5-7). If the host has sufficient residual resource capacity to host this VM, then we calculate the utilization variation of the utilization curve after allocating this VM during time period $[T'_0, T'_0+T]$ using Equ. (6) (Lines 9-10). Finally, we choose the PM that leads to the minimum utilization variation (Lines 11-15). It means this VM can make this PM have similar resource utilization between time epochs, and hence have the potential to host more VMs in the future and fully utilizes its resources.

3) Correlation Coefficient Based Mechanism: The utilization variation based algorithm ensures that the PM resource utilization does not spread out around the mean. However, it cannot fully reflect the complementariness of the VM utilization and PM utilization during the time period. In the example shown in Figure 12, we need to select a PM from two PMs to

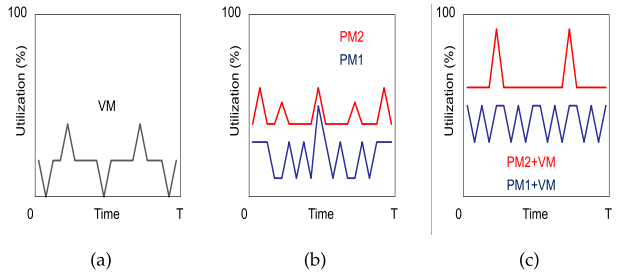


Fig. 12. Placing VM to PM1 and PM2. (a) VM. (b) PM1 and PM2. (c) PM1 + VM, PM2 + VM.

allocate a VM. The utilizations of the VM and PMs are shown in Figure 12(a) and Figure 12(b), respectively. Figure 12(c) shows the resource utilization after allocating the VM to each PM. Both curves have the same utilization variation value, so the two PMs are equivalent in PM selection since selecting either one will result in the same utilization variation according to Algorithm 3. However, PM1 is a better choice because it is more complementary to the VM, and will result in a more flat resource utilization, which enables to allocation more VMs in the PM. We then further propose a correlation coefficient based initial VM allocation mechanism.

We calculate the statistical correlation coefficient (denoted by c_r) for a VM with predicted resource demands $\mathcal{P}(t)$ and a PM j with residual resource capacity $\mathcal{R}(t)$ by (7), as shown at the bottom of this page, where $P^k(t_i)$ is predicted type-k resource demand at time t_i and $R_j^k(t_i)$ is residual type-k resource at time t_i, $\overline{P^k(t_i)}$ and $\overline{R_j^k(t_i)}$ are the average value. Therefore, for a VM, we aim to find a PM that has a correlation coefficient most close to -1 (i.e., the smallest correlation coefficient meaning the two traces are opposite to each other in terms of magnitude) with the VM as the destination PM to allocate this VM. Accordingly, we propose to select PMs based on the correlation coefficient of the VM utilization and PM utilization. As the VMs consume multiple types of resources, the algorithm first calculates the correlation coefficient for each resource and then calculates the average of all the correlation coefficients of different resources. The algorithm finds the PM that has the smallest average correlation coefficient (i.e., most close to -1) with the VM to be allocated as the VM's host. A PM with the smallest average correlation coefficient with the VM means that this VM allocation will result in resource utilization that does not fluctuate severely and hence has higher probability to accommodate more VMs.

Algorithm 4 shows the pseudocode for the correlation coefficient based VM allocation mechanism. Similar to Algorithm 2, this mechanism refers to the resource demand pattern $\mathcal{P}_i(t)$ and the residual resource capacity of candidates $\mathcal{R}_j(t)$. The algorithm first checks whether the candidate host has enough resource (Lines 5-7). It then calculates the correlation coefficient of the VM utilization and the residual resource capacity of the candidates for each type of resource based on Equ. (7) (Line 9). Specifically, the algorithm calculates the correlation coefficient values that are obtained from the utilization traces, and then selects the PM that has the smallest

Algorithm 4 Pseudocode for the Correlation Coefficient Based VM Allocation Mechanism

1: **Input:** $\mathcal{P}_i(t)$: Predicted resource demands
$\mathcal{R}_j(t)$: Residual resource capacity of candidates
2: **Output:** Allocated host of the VM. It is Null if it cannot be found.
3: AllocatedHost=Null
4: Cor=Double.MAX_VALUE
5: **for** $j = 1$ to m **do**
6: **if** CheckValid($\mathcal{P}(t)$,$\mathcal{R}_j(t)$)==**false then**
7: **continue**
8: **else**
9: $c_r = \frac{\sum_{k=1}^{d}\sum_{i=1}^{T}(P^k(t_i)-\overline{P^k(t_i)})(R_j^k(t_i)-\overline{R_j^k(t_i)})}{\sqrt{\sum_{k=1}^{d}\sum_{i=1}^{T}(P^k(t_i)-\overline{P^k(t_i)})^2\cdot\sum_{k=1}^{d}\sum_{i=1}^{T}(R_j^k(t_i)-\overline{R_j^k(t_i)})^2}}$
10: **end for**
11: **if** c_r<Cor **then**
12: Cor=c_r
13: AllocatedHost=host j
14: **end for**
15: **return** AllocatedHost

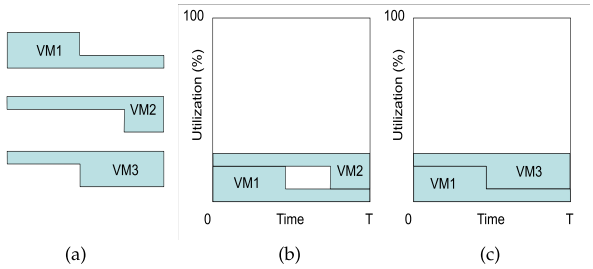


Fig. 13. Placing VMs vs. placing the VM group to PM. (a) VMs. (b) VM1 + VM2. (c) VM1 + VM3.

correlation coefficient (Lines 11-15). It means that this PM is the most complimentary to the VM across different resource, and allocating the VM to this PM can make this PM have similar resource utilization between time epochs during time period T. In this way, the algorithm is actually allocating complementary VMs (e.g., the VM is complementary with the existing VMs in the PM) to the same PM. As a result, the PM will have similar resource utilization between time epochs, and hence has potential to host more VMs in the future and fully utilizes its resources (i.e., more accommodating).

4) VM Group Based Mechanism: Rather than considering one VM, in this section, we try to place complementary VMs together by combining complementary VMs into a group first and then assigning the whole group to a PM. Compared to allocating VMs individually, combining complementary VMs into a group first for allocation has the advantage of extensively exploring the complementariness of the VMs and maximally consolidating complementary VMs, and hence can reduce the number of PMs needed. For example, suppose we allocate three VMs in the sequence of VM1, VM2 and VM3, as shown in Figure 13(a). Since the allocation result depends

$$c_r = \frac{\sum_{k=1}^{d}\sum_{i=1}^{T}(P^k(t_i) - \overline{P^k(t_i)})(R_j^k(t_i) - \overline{R_j^k(t_i)})}{\sqrt{\sum_{k=1}^{d}\sum_{i=1}^{T}(P^k(t_i) - \overline{P^k(t_i)})^2 \cdot \sum_{k=1}^{d}\sum_{i=1}^{T}(R_j^k(t_i) - \overline{R_j^k(t_i)})^2}} \tag{7}$$

on the allocating order of the VMs, Algorithm 4 will end up with placing VM1 and VM2 together as shown in Figure 13(b). However, VM3 is more complementary than VM2 to VM1. Placing VM3 and VM1 together is more preferred because it will result in similar resource utilization between time epochs in the PM, and hence make the PM more accommodating to other VMs. If we group complementary VMs together and then do the allocation, we can place VM1 and VM3 in one PM as shown in Figure 13(c) and hence make the PM more accommodating.

A question in grouping complementary VMs is which VM we should start with. In online VM allocation algorithms, it is difficult to find a PM to place a VM with high resource utilization variations, especially when such VMs are allocated later with less residual resources in PMs. Therefore, we give higher priorities to the VMs with higher utilization variation to start with in VM grouping, so that they will have more chances in finding complementary VMs. Specifically, in order to group complementary VMs together, we first sort the VMs based on the utilization variation in descending order. Then, we start from the first VM for VM grouping.

We can combine arbitrary number of VMs into one group, as long as the group resource demand does not exceed the PM resource capacity. We define the group resource demand as the combined resource demands of each type of resource of the VMs in the group. There is a tradeoff between the number of VMs that are selected to form a group and the complexity of the algorithm. In order to demonstrate the effectiveness of the VM group based mechanism and also achieve time efficiency of the mechanism, we combine two VMs in a group without the loss of generality. The procedure of combining VMs to groups is as follows. For each VM, we select the VM that is most complementary to it, and then combine these two VMs. For example, we calculate the correlation coefficients of this VM with all remaining VMs and select the one with the smallest correlation coefficient value. After that, we denote the VM groups as $G_n(n = 1, 2, \ldots)$, and sort the groups based on the group resource demand. Similar to the predicted resource demand pattern $\mathcal{P}_i(t)$ of a VM, the group resource demand is a d-dimension vector with each dimension representing its demands in one resource type. Suppose a group G_n comprises of m VMs, the combined resource demand of this group is:

$$\mathcal{P}_{G_n}(t) = (\sum_{i=1}^{m'} P_i^1(t), \sum_{i=1}^{m'} P_i^2(t), \ldots, \sum_{i=1}^{m'} P_i^d(t)) \quad (8)$$

where $P_i^k(t)$ is the type-k resource utilization of VM i, and $\sum_{i=1}^{m'} P_i^k(t)$ is the combined type-k resource demands of the m' VMs in the group. The group resource demand can be calculated by

$$S_{G_n} = \sum_{k=1}^{d} \left\{ w_k \frac{1}{T} \sum_{t=T_0'}^{T_0'+T} [\sum_{i=1}^{m'} P_i^k(t)]dt \right\}^2, \quad (9)$$

where $\frac{1}{T}\sum_{t=T_0'}^{T_0'+T} [\sum_{i=1}^{m'} P_i^k(t)]dt$ is the demand of type-k resource of group G_n, and w_k is the weight associated to type-k resource as in Equ. (5).

The reason for sorting the groups is that it is more difficult to find destination PMs to allocate the groups with large group resource demands, especially if such a group is allocated later after many other VM groups with few PM options left. Similar as the first-fit decreasing algorithm [31] that allocates large demand VM first, this algorithm can lead to fewer PMs used by allocating the groups with larger group resource demands first.

Similarly, we define the residual resource capacity of a PM based on the normalized residual resource capacity of the PM $\mathcal{R}_j(t) = (R_j^1(t), R_j^2(t), \ldots, R_j^d(t))$. The residual resource capacity of PM j is a positive scalar value representing the magnitude of the resource utilization in multiple dimensions, which can be calculated by

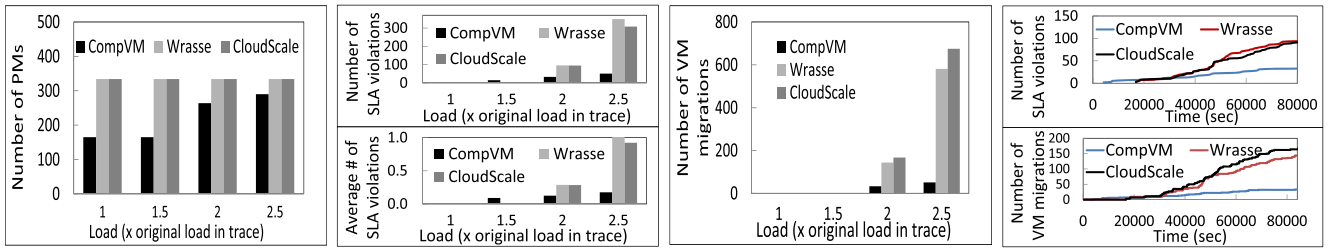
$$S_j = \sum_{k=1}^{d} \{ w_k \frac{1}{T} \sum_{t=T_0'}^{T_0'+T} R_i^k(t)dt \}^2, \quad (10)$$

where $\frac{1}{T}\sum_{t=T_0'}^{T_0'+T} R_i^k(t)dt$ is the residual resource capacity of type-k resource in the PM j; w_k is the assigned weight to resource k (the same with Equ. (5)).

Algorithm 5 shows the pseudocode for the VM group based allocation mechanism, that is used to derive the decisions of assigning VM groups to PMs, based on the residual resource capacities of PMs and group resource demands of VM groups. Given a list of VMs $\mathbb{L}_{VM}$ with their predicted resource demands $\mathcal{P}_i(t)$, and a list of PMs $\mathbb{L}_{PM}$ with their residual resource capacities $\mathcal{R}_j(t)$ (Line 1), the algorithm sorts the VMs in the descending order of their utilization variations calculated by Equ. (6) (Line 3). For each VM in the list $\mathbb{L}_{VM}$, the algorithm finds a VM that is the most complementary to the first VM (Lines 6-10), combines them into a group (Line 11), and then adds to the group list $\mathbb{L}_G$ (Line 12). The algorithm computes and sorts the groups based on their

Algorithm 5 Pseudocode for the VM Group Based Allocation Mechanism

```
1: Input: L_VM: list of VMs with predicted resource demands P_i(t)
          L_PM: list of PMs with residual resource capacities R_j(t)
2: Output: VM to PM mapping
3:     Arrays.sort(L_VM) //sorts the VMs in the descending order of utilization variations
4:     L_G = new Array()
5:     while L_VM not empty do
6:         VM1=L_VM.remove() // Removed VM from list
7:         for VM2 in L_VM
8:             Compute correlation coefficient of VM1 and VM2
9:         VM2=VM that has lowest correlation coefficient with VM1
10:        Remove VM2 from L_VM
11:        Create group G that comprises VM1 and VM2
12:        L_G.add(G)
13:    Compute group resource demands and residual resource capacities based on Equ. (9) and (10)
14:    while L_G is not empty do
15:        The biggest group G → the smallest feasible PM
16:        Remove the biggest group G from L_G
17:        if cannot find feasible PM then
18:            return False
19:    return VM to PM mapping
```



(a) (b) (c) (d)

Fig. 14. Performance under different workloads using the PlanetLab Trace. (a) The number of PMs used. (b) Total/average # of SLA violations. (c) The number of VM migrations. (d) # of SLA violations and migrations.

group resource demands and sort the PMs based on their residual resource capacities (Line 13), and then allocates the group with the biggest group resource demand to a feasible PM with the smallest residual resource capacity (Line 15). If a feasible PM cannot be found, the algorithm returns false (Lines 16-17), otherwise, it returns the VM-to-PM mapping after all the groups are allocated to the PMs (Line 18).

Note that it is possible that two VMs can be individually placed on PMs but when combined to a group, they do not fit into any PM. When a combined group cannot find a PM to host it, we will decompose the group to individual VMs and allocate each VM to a PM. Finally, we can combine all of these advanced algorithms in the VM allocation mechanism. First, the VMs are combined into groups. Then, correlation coefficient and utilization variation can be concurrently considered when selecting a PM to host a VM group. Different weights can be used for the metrics to give different priorities to them.

IV. Trace-Driven Simulation Performance Evaluation

In this section, we conducted the simulation experiments to evaluate the performance of CompVM using VM utilization trace from PlanetLab [21] and Google Cluster [22]. We used workload records of three days from the trace to generate VM resource request patterns and then executed CompVM for the fourth day's resource requests. We randomly selected the jobs and tasks from the traces. The window size was set to 15 in the pattern detection in CompVM. Note that our resource utilization prediction does not have 100% accuracy, and the actual resource demands may be higher than the predicted values, which leads to SLA violations and VM migrations. A higher prediction error leads to more SLA violations and VM migrations and vice versa. For more details of the prediction error, please refer to our publication in [32]. We compared CompVM with Wrasse [33] and CloudScale [34], which are dynamic VM allocation methods. All three methods first conduct initial VM allocation and then periodically execute VM migration by migrating VMs from overloaded PMs to first-fit PMs every 5 minutes. In the initial VM allocation, Wrasse and CloudScale place each VM to the first-fit PM based on the expected VM resource demands. Note that the expected demands are usually set to certain percentages of the peak demands (e.g., 80%). As a result, SLA violations can occur in the future due to VM workload fluctuation. In the VM migration step, when a PM becomes overloaded, it migrates out its VMs until it is no longer overloaded. The destination PM for each migration VM is the first-fit PM (i.e., the PM that has enough capacity to host the VM) in the PM list in the system. In VM migration, CloudScale first predicts future demands at a future time point and then migrates VMs to achieve load balance in the future time point. Next, we compared CompVM with enhancement mechanisms (proposed in Section III-B) with CompVM in order to show the effectiveness of the enhancement mechanisms.

In the default setup, we configured the PMs in the system with capacities of 1.5GHz CPU and 1536 MB memory and configured VMs with capacities of 0.5GHz CPU and 512 MB memory. With our experiment settings, the bandwidth consumption did not overload PMs due to their high network bandwidth capacities, so we focus on CPU and memory utilization. Unless otherwise specified, the number of VMs was set to 2000 and each VM's workload is twice of its original workload in the trace. We measured the following metrics after the simulation was run for 24 hours to report.

- *The number of PMs used.* This metric measures the energy efficiency of VM allocation mechanisms.
- *The number of SLA violations.* This is the number of occurrences that a VM cannot receive the required amount of resource from its host PM.
- *Average number of SLA violations.* This is the average number of SLA violations per PM. It reflects the effect of consolidating VMs into relatively fewer PMs.
- *The number of VM migrations.* This metric presents the cost of the allocation mechanisms that required satisfying VM demands and avoiding SLA violations. Note this paper only handles the VM initial allocation problem and does not handle the problem of VM migration. A better VM initial allocation algorithm will lead to fewer VM migrations. Therefore, we measured this metric just in order to show the performance of VM initial allocation.

A. Performance With Varying Workload

Figure 14 and Figure 15 show the performance of the three methods under different VM workloads using the PlanetLab trace and Google Cluster trace, respectively. We varied the workload of the VMs through increasing the original workload in the trace by 1.5, 2 and 2.5 times.

Figure 14(a) and Figure 15(a) show the total number of PMs used, which follows CompVM<CloudScale=Wrasse. CloudScale and Wrasse aim to avoid overloading each PM in initial VM placement and subsequent VM migration at each time point. This may result in some PMs that fully utilize one resource but under-utilize other resources, failing to fully utilize all resources. In contrast, in initial VM placement, CompVM consolidates complementary VMs in different resource dimensions, thus fully utilizing each resource in each PM. Since it considers the resource periodical utilization patterns during a certain time period, it reduces the VM migrations

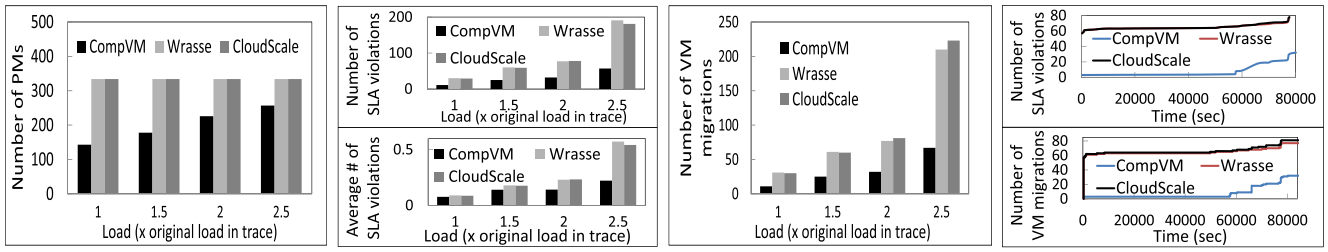


Fig. 15. Performance under different workloads using the Google Cluster Trace. (a) The number of PMs used. (b) Total/average # of SLA violations. (c) The number of VM migrations. (d) # of SLA violations and migrations.

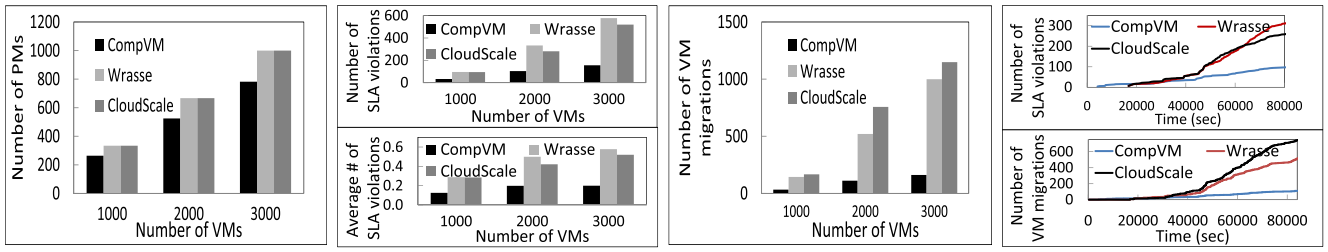


Fig. 16. Performance with different number of VMs using the PlanetLab Trace. (a) The number of PMs used. (b) Total/average # of SLA violations. (c) The number of VM migrations. (d) # of SLA violations and migrations.

and constrains the number of PMs used. Both figures also show that as the workload increases, the number of PMs of CompVM increases, while those of Wrasse and CloudScale remain the same. This is because as the actual workload increases, CompVM's predicted resource demands increase in initial VM placement, while CloudScale and Wrasse still allocate VM according to the labeled VM capacities.

Figure 14(b) and Figure 15(b) show the total number of SLA violations and the average number of SLA violations. We see that with the PlanetLab trace, when the workload is low, all three methods can provide service without violating SLAs. Both figures show that as the workload increases, both metric results increase and they exhibit CompVM<CloudScale<Wrasse. CompVM has fewer SLA violations because its predicted patterns can capture the time-varying VM resource demands and hence guarantee the resource provisioning. CloudScale has fewer SLA violations than Wrasse since CloudScale iteratively predicts VM resource demands and proactively migrates VMs before SLA violations occur. These results illustrate that CompVM maintains a smaller average number of SLA violations per PM even though it uses fewer PMs than CloudScale and Wrasse, which confirms CompVM's higher performance in energy efficiency and SLA guarantees.

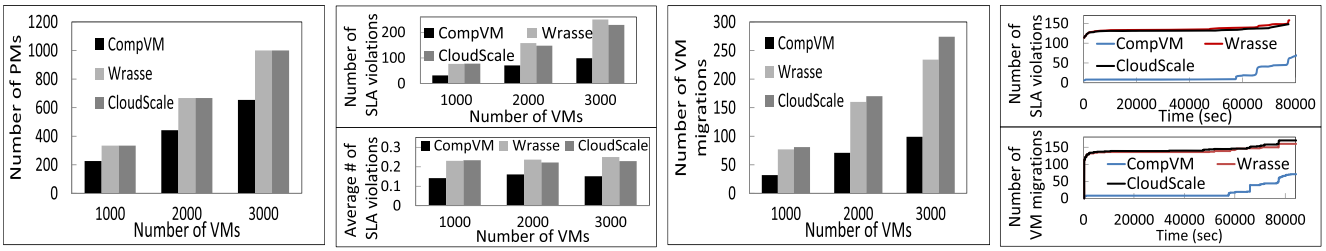
Figure 14(c) and Figure 15(c) show the total number of VM migrations in the three methods. Since the workload in the PlanetLab trace is relatively low compared to the Google Trace trace, when the workload is low, there are no SLA violations hence no VM migrations. Both figures show that as the workload increases, the number of VM migrations increases due to the increase of SLA violations as shown in Figure 14(b) and Figure 15(b). CompVM always triggers significantly fewer VM migrations than CloudScale and Wrasse due to its much fewer SLA violations. This experimental result confirms the effectiveness of CompVM in reducing VM migrations.

Figure 14(d) and Figure 15(d) show the accumulated number of SLA violations and VM migrations over time, respectively. In Figure 14(d), as the workload is low relative to PM capacity initially in the PlanetLab trace, all three methods have similar number VM violations and migrations at the early stage of simulation. As time goes on, due to the awareness of future resource demand pattern of the VMs during initial VM allocation, CompVM produces fewer VM violations and migrations than Wrasse and CloudScale during the experiment.

In the Google Cluster trace, the workload is high relative to PM capacity initially. Therefore, in Figure 15(d), due to the unawareness of future VM resource demands, the initial VM placement of Wrasse and CloudScale leads to around 60 VM migrations to guarantee enough resource provisioning. In contrast, CompVM generates 0 SLA violations and 0 migrations until at 6000s when the workload becomes higher. Recall that this experiment is for allocating one day's resource requests. Initially, all PMs do not have any VMs, and as time goes on, there are more VMs being hosted in a PM. At the beginning, due to CompVM's VM allocation strategy and low resource demands in the system, PMs are not likely to be overloaded. When the number of VMs in a PM is high to a certain extent, VMs' resource demand variance leads to PM overload, which triggers VM migrations. Therefore, after around 60000 seconds, the number of VM migrations in CompVM increases dramatically. The results in this figure are quite different with those in Figure 14(d), which has a higher number. As Section II-B indicates that the pattern of Google Cluster trace is more obvious than PlanetLab trace and hence is more suitable for pattern prediction by CompVM. We also observe that when the workload is high relative to PM capacity, most of the migrations are caused by inappropriate initial VM placement. Therefore, CompVM is significant in helping greatly reduce the SLA violations and VM migrations.

B. Performance With Varying Number of VMs

Figure 16 and Figure 17 present the performance of the three methods when the number of VMs was varied from 1000 to 3000 using the PlanetLab trace and the Google Cluster trace, respectively.



(a) (b) (c) (d)

Fig. 17. Performance with different number of VMs using the Google Cluster Trace. (a) The number of PMs used. (b) Total/average # of SLA violations. (c) The number of VM migrations. (d) # of SLA violations and migrations.

Figure 16(a) and Figure 17(a) show the total number of PMs used to provide service for the corresponding number of VMs. We see the result follows CompVM<CloudScale=Wrasse due to the same reasons as in Figure 14(a) and Figure 15(a). Also, as the number of VMs increases, the number of PMs used increases in each method since more PMs are needed to host more VMs. These experimental results confirm the advantage of CompVM in reducing the number of PMs used hence achieving higher energy efficiency.

Figure 16(b) and Figure 17(b) show the number of SLA violations and the average number of SLA violations per PM. We see both metric results follow CompVM < CloudScale<Wrasse due to the same reasons in Figure 14(b) and Figure 15(b). Also, as the number of VMs increases, both metric values in each method increase since more resource demands from more VMs lead to more SLA violations.

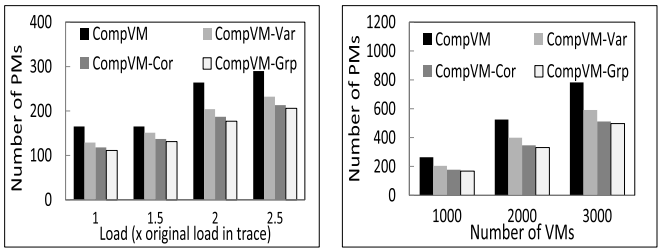
Figure 16(c) and Figure 17(c) show the total number of VM migrations in the three methods. As the number of VMs increases, the number of VM migrations increases due to the increase of SLA violations. CompVM always triggers significantly fewer VM migrations than CloudScale and Wrasse due to its much fewer SLA violations as shown in Figure 16(b) and Figure 17(b). CloudScale has slightly more migrations than Wrasse because it triggers VM migrations upon a predicted SLA violation, which may not actually occur. These experimental results confirm the effectiveness of CompVM in reducing VM migrations.

Figure 16(d) and Figure 17(d) show the number of migrations and SLA violations over time. The figures show similar trends of the three method as those shown in Figure 14(d) and Figure 15(d) due to the same reasons.

C. Performance of Enhancement Mechanisms

We implemented the improved initial VM allocation mechanisms described in Section III-B, and then compared them with CompVM, the original heuristic algorithm based on the average resource efficiency. We denote the utilization variation based mechanism as CompVM-Var (Algorithm 3), denote the correlation coefficient based mechanism as CompVM-Cor (Algorithm 4), and denote the VM group based mechanism as CompVM-Grp (Algorithm 5). Because the enhancement is only to more fully utilize PM resources while still ensuring that the VM demands will be satisfied during a period of time based on the demand prediction, we only measure the number of PMs to reflect the effectiveness of the enhancement.

Figure 18 compares the performance of the improved initial VM allocation mechanisms with CompVM using the PlanetLab trace. Figure 18(a) and Figure 18(b) show the total number of PMs used, with varying workloads and varying number of VMs, respectively. In both figures, the number



(a) (b)

Fig. 18. The number of PMs used with PlanetLab trace. (a) Varying workload. (b) Varying number of VMs.

of PMs follows CompVM-Grp≈CompVM-Cor<CompVM-Var<CompVM. CompVM-Var consolidates VMs to PMs based on the variation of resource utilization of PM after accommodating the VM. Compared to CompVM that is based on the average resource efficiency, CompVM-Var reduces the number of used PMs due to the reason that CompVM-Var tries to improve resource utilization by ensuring that the PM resource utilization does not spread out around the mean, and hence is able to consolidate more VMs, which leads to fewer PMs needed to host the VMs. CompVM-Cor further reduces the number of used PMs because the correlation coefficient based method ensures that the selected PM has the most complementary resource utilization to the VM and hence results in a high resource utilization during every time epoch, thus enabling a PM to host more VMs. CompVM-Grp has similar number of PMs to CompVM-Cor since both of them use correlation coefficient for mapping VMs to PMs.

Compared to CompVM-Cor, CompVM-Grp has a slightly fewer PMs because it considers a group of VMs rather than a single VM when assigning the VMs to the PMs. Combining complementary VMs into groups before allocating them to PMs has the advantage of extensively exploring the complementariness of the VMs and maximally consolidating complementary VMs, and hence enables a PM to host more VMs, which further reduces the total number of PMs needed to host all VMs. We also see that Figure 18(a) shows that as the workload increases, the number of PMs of all methods increases. This is because as the actual workload increases, the predicted resource demands increase in initial VM placement. Figure 18(b) shows that as the number of VMs increases, the number of PMs used increases in each method since more PMs are needed to host more VMs. These experimental results confirm the effectiveness of the improved initial VM allocation mechanisms in reducing the number of used PMs.

Similarly, Figure 19 compares the performance of the improved initial VM allocation mechanisms with CompVM using the Google Cluster trace. Figure 19(a) and Figure 19(b) show the total number of PMs used, with varying workloads

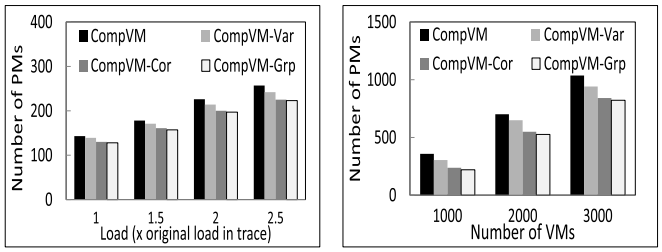


Fig. 19. The number of PMs used with Google Cluster trace. (a) Varying workload. (b) Varying number of VMs.

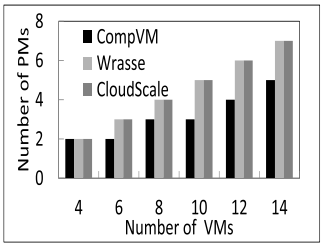


Fig. 20. # of PMs in testbed.

and varying number of VMs, respectively. We see that the number of PMs follows CompVM-Grp≈CompVM-Cor<CompVM-Var<CompVM, which is consistent with previous result using the PlanetLab trace due to the same reasons mentioned before. Using the Google Cluster trace, the improved initial VM allocation mechanisms do not reduce as many PMs as using the PlanetLab trace in the previous experiment. This is because the resource utilization from the Google Cluster trace does not fluctuate as severely as the resource utilization in the PlanetLab trace, hence the average resource efficiency mechanism performs well in guiding initial VM allocation. These experimental results again confirm effectiveness of the improved initial VM allocation mechanisms. The results also indicate that these mechanisms are more effective when the resource utilizations exhibit greater fluctuation.

V. Real-World Testbed Experiments

We deployed a real-world testbed to conduct experiments to validate the performance of CompVM in comparison with Wrasse and CloudScale. The testbed consists of 7 PMs (2.00GHz Intel(R) Core(TM)2 CPU, 2GB memory, 60GB HDD) and an NFS (Network File System) server with storage capacity of 80GB. We implemented CompVM, Wrasse and CloudScale in Java using the XenAPI library [35] running in a management PM (3.00GHz Intel(R) Core(TM)2 CPU, 4GB memory). We used the VM template of XenServer to create VMs (1VCPU, 256MB memory, 8.0GB virtual disk, running Debian Squeeze 6.0) in the cluster. We used publicly available workload generator *lookbusy* [36] to generate VM workloads.

Figure 20 shows the number of PMs used to provide the service of different number of VMs. Since Wrasse and CloudScale are unable to predict workload at beginning, they both use the maximum request resource of the VMs for allocation and hence have similar results. We also monitored the number of SLA violations during the experiment period, and found that were no SLA violations in all three methods during the experiment. These experimental results confirm that CompVM is able to provide service with fewer number of PMs than Wrasse and CloudScale.

TABLE I
VM Allocation Mapping

PM	CompVM	Wrasse	CloudScale
PM1	VM1, VM2	VM1, VM2	VM1, VM2
PM2	VM3, VM4, VM5	VM3, VM4	VM3, VM4
PM3	-	VM5	VM5

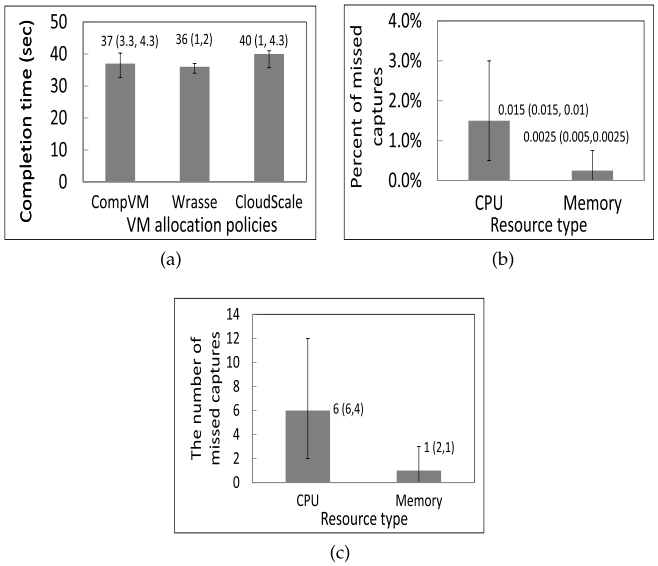


Fig. 21. Performance of running WordCount job on the real-world testbed. (a) Job completion time. (b) % of missed captures in CompVM. (c) # of missed captures in CompVM.

We then deployed a virtual cluster with 5 VMs collaboratively running the WordCount Hadoop benchmark job. We first conducted a profiling run of such MapReduce job and used the collected resource utilization to generate patterns for initial VM allocation in CompVM. The 5 VMs were initially allocated to different PMs by different methods. The initial VM to PM mapping is shown in Table I. We see that CompVM uses fewer PMs than Wrasse and CloudScale. During the experiment, no SLA violations were detected in all three methods. Figure 21(a) shows the median, 10th percentile and 90th percentile of the job completion time in ten experiments. We see that though CompVM uses few PMs, it has a similar completion time as Wrasse and CloudScale. This result verifies the advantage of CompVM in requiring fewer PMs without sacrificing the performance quality of the VMs.

Figure 21(b) shows the median, 10th and 90th percentiles of the percent of missed captures of CompVM during the experiment. Figure 21(c) shows the median, 10th and 90th percentiles of the number of missed captures of CompVM during the experiment. We see that CompVM produces very few missed captures relative to the total number of predictions at each time point, which verifies the effectiveness of CompVM in resource demand pattern detection. We also see that the percent of missed captures of CPU and its variance are relatively larger than those of memory. This is due to the reason that the memory utilizations of the VMs exhibit more obvious patterns and hence are easier to be captured in pattern detection.

VI. Related Work

A significantly amount of attention has been paid to VM allocation strategies recently [37]. Accordingly, many static and dynamic VM allocation strategies have been

proposed [2]. Static provisioning [3]–[7] allocates physical resources to VMs only once based on static peak VM resource demands. However, static provisioning cannot fully utilize resources because of time-varying resource demands of VMs. To fully utilize cloud resources, dynamic provisioning [8]–[13] first consolidates VMs using a simple bin-packing heuristic and then use live VM migrations, which however results in high migration overhead.

Many VM placement methods have been proposed for different purposes. Ahvar *et al.* [38] proposed a cost and carbon emission-efficient VM placement method in distributed clouds. Hao *et al.* [39] studied where to provision and place a VM at the "right place" (i.e., data center, rack and host) so that both the current and future needs for the VM can be satisfied. Lin *et al.* [40] presented a VM placement algorithm based on the peak workload characteristics and measured the similarity of VMs' workload with VM peak similarity. Mann [41] investigated how mapping VMs to PMs (i.e., VM placement) or mapping computational tasks to VMs (i.e., VM selection) influence each other. Nejad *et al.* [42] formulated the dynamic VM provisioning and allocation problem for the auction-based model as an integer program and proposed solutions. Ruan and Chen [43] leveraged the performance-to-power ratios for various PM types to guarantee that PMs run at the most power-efficient levels so that the energy consumption can be tremendously reduced with little sacrifice of performance. Wang *et al.* [44] proposed a stable matching-based VM allocation mechanism to achieve better resource utilization and thermal distribution while satisfying SLA requirements. These strategies consider the current state of resource demand and available capacity at a time point rather than the trend state during a time period for VM migration, which is insufficient for maintaining a load balanced state for a long time. Our idea of consolidating complementary VMs for a certain time period can help these migration strategies maintain the load balanced state for a longer time period.

Some works [34], [45]–[49] predict resource demands for VM migration to avoid SLA violation in the future. All these VM allocation strategies consider the current or future state of resource demand and available capacity at a time point rather than during a time period, which is insufficient for maintaining a continuous load balanced state. Though our work focuses on initial VM allocation rather than subsequent VM migration, our idea of consolidating complementary VMs for a certain time period can help the migration strategies maintain the load balanced state for a longer time period.

Some previously proposed resource provisioning methods consider spatial balance (e.g., [15]–[18]) or temporal correlations (e.g., [19], [20]). Xiao *et al.* [15] introduced the concept of "skewness" to measure the unevenness in the multidimensional resource utilization of a server, and try to minimize skewness by combining different types of workloads to improve the utilization of server resources. He *et al.* [16] proposed an algorithm that uses a multivariate probabilistic model to select suitable PMs for VM re-allocation that can capture the multi-dimensional characteristics of VMs and PMs. Ni *et al.* [17] proposed a VM mapping policy based on multi-resource load balancing. It uses the resource consumption of the running VM and the self-adaptive weighted approach, which resolves the load balancing conflicts of each independent resource caused by different demand for resources of cloud applications. Mishra and Sahoo [18] proposed a VM allocation methodology based on vector arithmetic and build theory which can be used not only to make the decisions robust but also to make the process of choosing PMs easier and more appropriate. Verma *et al.* [19] presented detailed analysis of an enterprise server workload from the perspective of finding characteristics for consolidation. Ganesan *et al.* [20] proposed a tool iCirrus-WoP that determines VM capacity and VM collocation possibilities for a given set of application workloads.

VII. Conclusions

In this paper, we propose an initial VM allocation mechanism (called CompVM) for cloud datacenters that consolidates complementary VMs with spatial/temporal-awareness. This mechanism consolidates complementary VMs into one PM, so that in each dimension of the multi-dimensional resource space, the sum of the resource consumption of the VMs nearly reaches the capacity of the PM during the VM lifetimes. Specifically, given a requested VM, CompVM predicts the resource demand pattern of this VM, and then finds a PM that has a remaining resource pattern complement to the VM resource demand pattern, i.e., the PM has the least residual capacity in each resource dimension big enough to hold this VM. We propose an average resource efficiency based VM allocation mechanism to coordinate the requirements of different resources and consolidates complementary VMs in the same PM. We further propose an utilization variation based and a correlation coefficient based mechanisms to identify complementary VMs and allocate them to PMs. We also propose a VM group based mechanism that combines complementary VMs into groups before assigning. As a result, CompVM helps fully utilize the cloud resources, and reduce the number of PMs needed to satisfy tenant requests. It also reduces the numbers of subsequent VM migrations and SLA violations. These advantages have been verified by our extensive simulation experiments based on two real traces and real-world testbed experiments running a MapReduce job. In our future work, we will explore how to enhance the pattern detection method to catch peak bursts and how to complement VMs with peak bursts in resource consumption.

References

[1] B. Rajkumar, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generat. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[2] H. Viswanathan *et al.*, "Energy-aware application-centric VM allocation for HPC workloads," in *Proc. IPDPS Workshops*, 2011, pp. 890–897.

[3] U. Bellur, C. S. Rao, and S. M. Kumar. (Nov. 2010). "Optimal placement algorithms for virtual machines." [Online]. Available: https://arxiv.org/abs/1011.5064

[4] J. Xu and J. A. B. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. CPSCom*, 2010, pp. 179–188.

[5] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. SIGCOMM*, 2011, pp. 242–253.

[6] L. Popa *et al.*, "FairCloud: Sharing the network in cloud computing," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 187–198, 2012.

[7] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proc. HotPower*, 2008, pp. 1–5.

[8] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Comput. Netw.*, vol. 53, no. 17, pp. 2923–2938, 2009.

[9] M. Tarighi, S. A. Motamedi, and S. Sharifian. (Feb. 2010). "A new model for virtual machine migration in virtualized cluster server based on fuzzy decision making." [Online]. Available: https://arxiv.org/abs/1002.3329

[10] E. Arzuaga and D. R. Kaeli, "Quantifying load imbalance on virtualized enterprise servers," in *Proc. WOSP/SIPEW*, 2010, pp. 235–242.
[11] A. Singh, M. R. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *Proc. SC*, 2008, Art. no. 53.
[12] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *Proc. NOMS*, 2009, pp. 373–381.
[13] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Proc. Middleware*, 2008, pp. 243–264.
[14] H. Shen, Y. Zhu, and W.-N. Li, "Efficient and locality-aware resource management in wide-area distributed systems," in *Proc. NAS*, 2008, pp. 287–294.
[15] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.
[16] S. He, L. Guo, M. Ghanem, and Y. Guo, "Improving resource utilisation in the cloud environment using multivariate probabilistic models," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 574–581.
[17] J. Ni, Y. Huang, Z. Luan, J. Zhang, and D. Qian, "Virtual machine mapping policy based on load balancing in private cloud environment," in *Proc. Int. Conf. Cloud Serv. Comput.*, 2011, pp. 292–295.
[18] M. Mishra and A. Sahoo, "On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach," in *Proc. IEEE Int. Conf. Cloud Comput.*, Jul. 2011, pp. 275–282.
[19] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. USENIX Annu. Techn. Conf.*, 2009, pp. 355–368.
[20] R. Ganesan, S. Sarkar, and A. Narayan, "Analysis of SaaS business platform workloads for sizing and collocation," in *Proc. USENIX Annu. Techn. Conf.*, 2012, pp. 868–875.
[21] *PlanetLab Traces*. Accessed: Apr. 2017. [Online]. Available: https://github.com/beloglazov/planetlab-workload-traces
[22] *Google Cluster Data*. Accessed: Apr. 2017. [Online]. Available: https://code.google.com/p/googleclusterdata/
[23] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Proc. NOMS*, 2012, pp. 1287–1294.
[24] D. Xie, N. Ding, Y. C. Hu, and R. R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 199–210, 2012.
[25] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," *Comput. Netw.*, vol. 53, no. 17, pp. 2905–2922, 2009.
[26] J. Liu and H. Shen, "CORP: Cooperative opportunistic resource provisioning for short-lived jobs in cloud systems," in *Proc. CLUSTER*, 2016, pp. 90–99.
[27] C. Qiu, H. Shen, and L. Chen, "Probabilistic demand allocation for cloud service brokerage," in *Proc. INFOCOM*, 2016, pp. 1–9.
[28] J. Csirik, J. B. G. Frenk, M. Labbé, and S. Zhang, "On the multidimensional vector bin packing," *Acta Cybern.*, vol. 9, no. 4, pp. 361–369, 1990.
[29] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," Microsoft Res., Tech. Rep., 2010.
[30] L. Chen, H. Shen, and K. Sapra, "RIAL: Resource intensity aware load balancing in clouds," in *Proc. INFOCOM*, 2014, pp. 1294–1302.
[31] B. S. Baker, "A new proof for the first-fit decreasing bin-packing algorithm," *J. Algorithms*, vol. 6, no. 1, pp. 49–70, 1985.
[32] L. Chen and H. Shen, "Considering resource demand misalignments to reduce resource over-provisioning in cloud datacenters," in *Proc. INFOCOM*, 2017, pp. 1–9.
[33] A. Rai, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud," in *Proc. SOCC*, 2012, Art. no. 15.
[34] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: Elastic resource scaling for multi-tenant cloud systems," in *Proc. SOCC*, 2011, Art. no. 5.
[35] *Xenapi*. Accessed: Feb. 2015. [Online]. Available: http://community.citrix.com/display/xs/Download+ SDKs
[36] *Lookbusy*. Accessed: Apr. 2017. [Online]. Available: http://devin.com/lookbusy/
[37] Z. Á. Mann, "Allocation of virtual machines in cloud data centers—A survey of problem models and optimization algorithms," *ACM Comput. Surv.*, vol. 18, no. 1, 2015, Art. no. 11.
[38] E. Ahvar *et al.*, "CACEV: A cost and carbon emission-efficient virtual machine placement method for green distributed clouds," in *Proc. IEEE SCC*, Jun. 2016, pp. 275–282.
[39] F. Hao, M. Kodialam, and T. V. Lakshman, "Online allocation of virtual machines in a distributed cloud," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 238–249, Feb. 2017.
[40] W. Lin, S. Xu, J. Li, L. Xu, and Z. Peng, "Design and theoretical analysis of virtual machine placement algorithm based on peak workload characteristics," *Soft Comput.*, vol. 21, no. 5, pp. 1301–1314, 2017.
[41] Z. Á. Mann, "Interplay of virtual machine selection and virtual machine placement," in *Proc. ESOCC*, 2016, pp. 137–151.
[42] M. M. Nejad, L. Mashayekhy, and D. Grosu, "Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 2, pp. 594–603, Feb. 2015.
[43] X. Ruan and H. Chen, "Performance-to-power ratio aware virtual machine (VM) allocation in energy-efficient clouds," in *Proc. CLUSTER*, 2015, pp. 264–273.
[44] J. Wang, K. Fok, and C. Cheng, "A stable matching-based virtual machine allocation mechanism for cloud data centers," in *Proc. IEEE SERVICES*, Jun. 2016, pp. 103–106.
[45] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. IM*, 2007, pp. 119–128.
[46] L. Chen, H. Shen, and K. Sapra, "Distributed autonomous virtual resource management in datacenters using finite-Markov decision process," in *Proc. SOCC*, 2014, pp. 1–13.
[47] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proc. IEEE*, vol. 102, no. 1, pp. 11–31, Jan. 2014.
[48] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proc. ICDCS*, 2013, pp. 400–409.
[49] M. Verma, G. R. Gangadharan, N. C. Narendra, and R. Vadlamani, "Dynamic resource demand prediction and allocation in multi-tenant service clouds," in *Proc. CCPE*, 2016, p. 4429–4442.
[50] L. Chen and H. Shen, "Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters," in *Proc. INFOCOM*, 2014, pp. 1033–1041.

Haiying Shen (SM'13) received the B.S. degree in computer science and engineering from Tongji University, China, in 2000, and the M.S. and Ph.D. degrees in computer engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor with the Computer Science Department, University of Virginia. Her research interests include cloud computing and cyber-physical systems. She is a Microsoft Faculty Fellow of 2010 and a member of the ACM. She was the program co-chair for a number of international conferences and a member of the Program Committee of many leading conferences.

Liuhua Chen received the B.S. and M.S. degrees from Zhejiang University in 2008 and 2011, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Clemson University. His research interests include distributed and parallel computer systems and cloud computing.