PageRankVM: A PageRank Based Algorithm with Anti-Collocation Constraints for Virtual Machine Placement in Cloud Datacenters

Zhuozhao Li, Haiying Shen and Cole Miles
Department of Computer Science
University of Virginia
Email: {zl5uq, hs6ms, cm3uc}@virginia.edu

Abstract— **There is a dramatic increase in the variety of virtual machines (VMs) and complexity of VM placement problems in clouds. Previous VM placement approaches attempt to accommodate more VMs efficiently on fewer PMs by balancing the resource usages across multiple dimensions. However, these approaches are not sufficiently accurate in measuring the quality of the PMs in terms of fully utilizing PM resource and having the potential to accommodate more VMs. Therefore, it is critical to design a new method that can more accurately measure the probability of a PM of fully utilizing its resources after accommodating a given VM with the consideration of different types of VMs. In addition, anti-collocation constraints must be handled efficiently. We propose a PageRank based VM placement algorithm with anti-collocation constraints (PageRankVM). PageRankVM defines the best PM resource usage profile, which means that the PM has full resource utilization for every resource dimension, and then ranks PM profiles according to their convergence of transferring (by accommodating VMs) to the best profile. PageRankVM then places a given VM to the PM based on the ranks of the resulted PM profiles after accommodating the VM with the consideration of anti-collocation constraints. Compared to previous approaches, PageRankVM effectively measures the ability of different PM profiles to reach the best profiles by accommodating a given VM, and hence differentiates the effectiveness of different VM placement decisions. We conducted extensive trace-driven simulation and GENI testbed experiments and demonstrated that PageRankVM has superior performance compared with other methods in terms of reducing the number of PMs, the energy consumption, the number of VM migrations, and the service level objective (SLO) violations.**

I. Introduction

Cloud computing attracts many enterprises (e.g., Dropbox, Facebook video storage) to migrate their business or services to the clouds without the need to build their own datacenters. Cloud providers multiplex computation, storage and network resources [32], [33] among different customers, using virtualization technologies to allocate Physical Machine (PM) resources to customer Virtual Machines (VMs) based on their resource (e.g., CPU, memory and bandwidth) requirements. The VM placement that assigns a set of VMs requested by customers to PMs is an important datacenter resource management problems. The VM placement problem aims to achieve a certain objective, while subject to several constraints such as the PM resource capacity constraints in multiple resources types (i.e., dimensions) and VM collocation or anti-collocation constraints. *Anti-collocation* constraints mean that the requested resources of a VM are required not to be placed in the same physical hardware in order to improve parallelism and performance.

The multi-dimension VM placement problem is analogous to the multi-dimension bin packing problem, which assigns the balls to the least number of bins given the sizes of the balls and bins. Since the multi-dimension bin packing problem is known as NP-Hard [30], there exist several heuristics [27], [30], [10]. Practical cloud systems [27] usually adopt less complicated heuristics, such as round-robin [24], first-fit [27] and first-fit-decrease [30], as evidenced by open-source middleware stacks [15].

The first-fit method places a VM to first PM that has sufficient resources to host this VM. The first-fit-decrease method calculates the size of PM by weighted sum of the d-dimensional vector, which represents the resource capacity in d dimensions and greedily places VMs to PMs with decreasing sizes. The work in [10] allocates a VM to the best-fit PM that has the minimum remaining resources after allocating the VM. However, these approaches are not sufficiently accurate in measuring the quality of the PMs in terms of fully utilizing PM resource and having the potential to accommodate more VMs since they neglect that there are different types of VMs and a VM's demands on different resource dimensions may be unbalanced. Therefore, *it is critical to design a new method that can more accurately measure the probability of a PM of fully utilizing its resources after accommodating a given VM, considering the different types of VMs.*

In addition, system deployment requested by customers may contain complex relationships among VMs such as anti-collocation constraints. Anti-collocation is desirable due to many reasons. Most VM types offered by public clouds such as Amazon EC2 [14] have multiple virtual CPUs (vCPUs) and virtual disks per VM. When a customer requests a VM with multiple vCPUs, (s)he usually prefers that the vCPUs be placed in different physical CPU cores in order to improve the parallelism of the vCPUs. When a customer requests a VM with multiple virtual disks, (s)he may prefer that the VM's virtual disks be spread out across the physical disks of the PM so that the accesses to different virtual disks do not interfere with each other, and also the fault tolerance and availability can be improved.

However, the anti-collocation requirements have not been adequately addressed in the previous VM placement solutions. Biran *et al.* [7] and Xia *et al.* [36] used mixed integer programming (MIP) formulations and algorithms (e.g., integer programming software Gurobi [20]) for the VM placement problem with anti-collocation requirements. However, the MIP formulation has an exceedingly large number of variables and constraints. Standard MIP algorithms cannot solve large problem instances within an acceptable period of time. Furthermore, the algorithms [7], [36] only consider anti-collocation requirement for one resource (e.g., disk), which is not practical for problems that have multiple types of anti-collocation requirements (e.g., vCPU anti-collocation and disk anti-collocation), which otherwise would make solving the MIP formulation far more complex. *Therefore, it is desirable to develop a VM placement algorithm with low computational complexity and can effectively solve the VM placement problem with multiple anti-collocation requirements.*

To address the aforementioned two problems, we propose a PageRank [29] based heuristic algorithm to place VMs to PMs based on how likely it is that the PM can later get to full PM resource usage, called *PageRankVM*. We abstract the resource usage of a PM across multiple dimensions as a resource usage profile. To address the anti-collocation requirement, we consider each unit of resource such as each core or each disk as a resource dimension in a PM's profile. *The anti-collocation resources of the same type (e.g., several vCPUs) of a VM are also represented in different dimensions in a VM's resource request, but can be in different permutations.* PageRankVM defines the best PM profile, which means that the PM has full resource utilization for all resource dimensions, and then ranks PM profiles according to their convergence of transferring (by accommodating VMs) to the best profile. Compared to previous approaches, PageRankVM effectively and properly measures the ability of different PM profiles to reach the best profiles by accommodating a given VM, and hence differentiates the effectiveness of VM placement decisions. PageRankVM then places a given VM to the PM based on the ranking scores of the resulted PM profiles after accommodating the VM.

The PageRank algorithm was originally designed by Google to rank webpages in their search engine results. PageRank works by treating each backlink to a webpage as if it were a vote of support for that webpage, and by giving weight to the vote based on the rank of the voting (i.e., linking) webpage. The underlying assumption is that more important webpages are likely to receive more links from other webpages [29]. We build a PageRankVM graph that ranks each possible resource usage profile of PMs on a scale from least accommodating to most accommodating in terms of reaching the best PM profile. For example, a PM resource usage profile can be represented by the vector $\mathbf{p} = \{p_1, ..., p_i, \; ..., p_m\}$, where each dimension p_i represents the resource utilization in the corresponding dimension. If the resource usage profile $\mathbf{p}_A$ of a PM can change to profile $\mathbf{p}_B$ (i.e., resource usage status

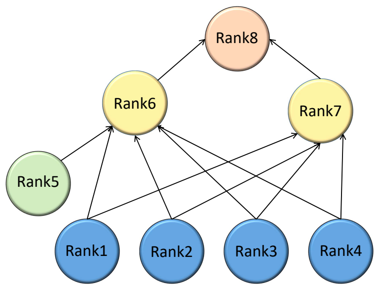


Fig. 1. PageRank graph showing the rank values of different PM profiles.

are changed) after accommodating a VM (among the existing VM types), we consider profile $\mathbf{p}_A$ to be a vote of support for profile $\mathbf{p}_B$. The vote from $\mathbf{p}_A$ is given a weight based on the rank of $\mathbf{p}_A$ itself. Figure 1 illustrates the PageRank graph showing the rank values of different PM profiles. For a given VM, its placement is performed by adding it to the PM that gives the highest rank for the new profile that is produced.

The main contributions of the paper include:

- We show that previous variance based approaches, which place VMs to the PMs that will result in the minimum variances of resource utilization across different dimensions, do not necessarily lead to high utilization of PM resources and reduce the number of PMs needed.
- We propose a PageRank score based algorithm to identify the preferred VM placement decisions (e.g., preferred resource utilization status of the PMs). We then leverage this proposed PageRank score based algorithm to solve the multiple dimensional VM placement problem.
- We implemented PageRankVM and conducted extensive experiments with trace-driven simulation and on the GENI testbed to show the advantages of PageRankVM.

The rest of this paper is organized as follows. Section II briefly presents the related work. Section III presents the motivation of our work. Section IV formulates the VM placement problem with resource anti-collocation requirements. Section V presents our PageRank based VM placement algorithm. Section VI presents the evaluation of PageRankVM in trace-driven simulation and real GENI testbed. Finally, Section VII summarizes the paper with remarks on our future work.

II. Related Work

Prior VM placement studies involve consolidating VMs, with multiple dimensional resource requirements, into the least number of servers. Most practical cloud systems [1], [28], [15] use simple and efficient heuristics such as first-fit [27] and first-fit-decrease [30], [34] to quickly find PMs for VMs. These approaches greedily place a VM to the PM that has sufficient resources for the VM. That is, they only check whether the requested CPU and memory of the VMs are satisfiable when assigning VMs to PMs, which however may result in resource fragmentation. Some previous works solve the VM placement problem based on the multi-dimensional bin packing algorithms [30], [5], [25], [13], [2], [19].

In order to solve the VM placement problem with complex multiple dimensions and anti-collocation constraints, many works [7], [36] advocate the use of mixed integer programming [17] formulations and algorithms. The work in [3] describes how datacenters can offer services with complex resource requirements to the cloud customers, such as entire IT as a service with VM collocation and anti-collocation requirements. Biran *et al.* [7] used the integer programming techniques to solve the VM placement problem that not only satisfy the predicted communication demand of the VMs but also the time varying demands. Xia *et al.* [36] formulated the assignment of VMs to PMs as an integer optimization problem that can capture constraints related to the available resources, datacenter efficiency and customers' complex requirements, and used a mixed integer programming (MIP) algorithm to solve the VM placement problem with anti-collocation requirements. However, as we mentioned in Section I, the MIP formulation has an exceedingly large number of variables and constraints and solving the MIP formulation with multiple types of anti-collocation requirements is more complex.

In VM migration, many important factors such as reducing power consumption and reducing resource wastage need to be considered. Chen *et al.* [12] proposed a resource intensity aware VM migration method to dynamically assign different weights to different resources according to their usage intensity in the PM, which reduces the time and cost to achieve load balance in VM migration. Chen *et al.* [11] proposed a proactive Markov Decision Process (MDP)-based VM migration algorithm that allows a PM to proactively find an optimal action on moving out a specific VM to transit to a lightly loaded state that will maintain for a longer period of time.

However, there has been few efforts devoting to studying how to consider the different types of VMs in VM placement to more fully utilize PM resources and how to efficiently solve the VM placement problem with multiple anti-collocation requirements. This paper addresses these two problems.

III. Motivation

A. PM Usage Profile

We abstract the resource usage status of a PM as a profile (i.e., resource utilization across different resource dimensions). In the following discussion, we present a PM profile in the form of $[p_1, ..., p_i, ..., p_m]$, where $i = 1, 2, ..., m$ represents different resource dimensions. For example, a profile $[p_1, p_2, p_3, p_4]$ can be used to represent a PM that has p_1, p_2, p_3 and p_4 usages in CPU, memory, network and disk, respectively. Also, we can use a profile to represent the usage of a CPU with different usages across its cores. In the following, we do not distinguish the actual types of resources represented by the dimensions. We focus on the mathematical characteristics of the dimensions. Similar idea is also applied to define different VM types. We present VM type in the form of $[v_1, ..., v_i, ..., v_m]$, where $i = 1, 2, ..., m$. $[v_1, ..., v_i, ..., v_m]$ represent the VM resource demand across multiple dimensions.

B. Limitations of Existing Approaches

When selecting a PM to accommodate a VM, existing approaches are mainly based on two requirements.

(1) The resource utilization of the corresponding PM u is maximized after hosting the VM.

(2) The variance of resource utilization across different dimensions v is minimized after hosting the VM.

Here, the resource utilization of a PM with the profile $[p_1, ..., p_m]$ is defined as $u = \sum_{i=1}^{m} p_i$, and the variance of resource utilization across different dimensions is defined as $v = \frac{1}{m} \sum_{i=1}^{m} (p_i - \frac{u}{m})^2$. The rationale of the first requirement is that the PM resource is fully used. The rationale of the second requirement is that the PM has the potential to accommodate more VMs.

However, we argue that *given a VM, even if a PM profile satisfies the aforementioned two requirements after it hosts a VM, it does not necessarily mean that it can host more VMs (i.e., a better host option) than a PM profile that does not satisfy the two requirements after it hosts the VM.*

We demonstrate this argument using an example. We assume the capacity of each dimension is 4. Specifically, we study the profiles $[0, 0, 0, 0] \backsim [4, 4, 4, 4]$. We use $[x, x]$ or $[x, x, x, x]$ to represent the resource demands of the VMs. $[x, x]$ means the resources can be allocated to any two dimensions and its permutations include $[x, x, 0, 0]$, $[0, 0, x, x]$, $[x, 0, 0, x]$ and so on. [1,1,1,1] represents the resource demands of the VM across 4 dimensions. We assume there are two VM types $\{[1, 1], [1, 1, 1, 1]\}$.

Suppose there are two PM options for a given VM, and their profiles become $[4, 3, 3, 3]$ and $[3, 3, 2, 2]$ after hosting this VM. $[4, 3, 3, 3]$ has utilization 13 and variance 0.75, and $[3, 3, 2, 2]$ has utilization 10 and variance 1. Although profile $[4, 3, 3, 3]$ has higher utilization and lower variance than profile $[3, 3, 2, 2]$, profile $[4, 3, 3, 3]$ cannot be considered as a better profile (that can accommodate more VMs) after hosting the VM when compared to profile $[3, 3, 2, 2]$. This is because it is impossible for $[4, 3, 3, 3]$ to develop to the best profile $[4, 4, 4, 4]$ by hosting more VMs, while there are multiple ways for $[3, 3, 2, 2]$ to develop to the best profile; accommodating one $[1, 1, 1, 1]$ VM and one $[1, 1]$ (i.e., $[0, 0, 1, 1]$) VM, and accommodating three $[1, 1]$ VMs (i.e., $[1, 1, 0, 0]$, $[0, 0, 1, 1]$ and $[0, 0, 1, 1]$).

Therefore, *it is critical to design a new algorithm that can more accurately measure the probability of a PM of fully utilizing its resources after accommodating a given VM, considering the various types of VMs.*

IV. Analytic Model

The initial VM allocation must consider multi-dimensional resource demands of actual VMs, which may or may not be balanced. Our primary goal is to consider the various types of VMs in VM placement in order to fully utilize the resources of each PM and hence minimize the number of PMs needed.

Consider that a datacenter has M PMs. Cloud users request N VMs with resource requirements across multiple dimensions and resource anti-collocation constraints. Usually, the value of N and M can be fairly large in a cloud datacenter, ranging from tens of thousands to hundreds of thousands. VM requests can be different in resource requirements on the number of vCPUs, memory, the number of local disks, and respective volume sizes of the disks. PMs are heterogeneous and provide certain resource capacities in the form of the number of vCPUs, memory, the number of local disks and their respective volume sizes.

Notations. Let the sets of VMs and PMs be denoted by $\mathcal{V}$ and $\mathcal{P}$, respectively. Without loss of generality, let $\mathcal{V} = \{V_1, V_2, ..., V_N\}$ and $\mathcal{P} = \{P_1, P_2, ..., P_M\}$, where V_i and P_j denote VM i and PM j, respectively. To address the anti-collocation requirement, we consider each unit of resource such as each core or each disk as a resource dimension in a PM's profile. For example, originally we consider *all* physical CPU cores in one PM as one dimension of resource. Now, we consider *each* physical CPU core in a PM as one dimension of resource. The anti-collocation resources of the same type (e.g., several vCPUs) of a VM are also represented in different dimensions in a VM's resource request, but can be in different permutations. For each VM i, let $\mathbf{c}_i$=$\{\alpha_i^1, \alpha_i^2, ..., \alpha_i^{|\mathbf{c}_i|}\}$ be a set of the vCPUs requested by the VM, $|\mathbf{c}_i|$ is the number of requested vCPUs. For each of the CPU cores, α_i^k represents the requested amount of CPU capacity in gigahertz (GHz). Usually for a VM request, we have $\alpha_i^1 = \alpha_i^2 = ... = \alpha_i^{|\mathbf{c}_i|}$. Notice that this resource demand is permutable. For example, $\{\alpha_i^1, \alpha_i^2, 0, 0\}$ and $\{0, 0, \alpha_i^1, \alpha_i^2\}$ are essentially the same resource demand. This is because the VM only requires resources from two cores, but does not care which cores each vCPU is assigned to run. Let β_i be the memory requirement in gibibyte (GiB). For each VM i, let $\mathbf{d}_i = \{\gamma_i^1, \gamma_i^2, ..., \gamma_i^{|\mathbf{d}_i|}\}$ be the set of virtual disks requested, where $|\mathbf{d}_i|$ is the number of requested virtual disks. For each of the virtual disks, γ_i^k represents the requested disk volume size in gigabyte (GB). Thus, the resource request of VM i can be represented by $\mathbf{r}_i = \{\mathbf{c}_i, \beta_i, \mathbf{d}_i\}$. Similarly, for each PM j, let $\mathbf{C}_j = \{A_j^1, A_j^2, ..., A_j^{|\mathbf{C}_j|}\}$ be the set of physical CPU cores it can support, where $|\mathbf{C}_j|$ is the number of CPU cores. For each CPU core, A_i^k represents the compute capacity in gigahertz (GHz). Let B_j be the amount of memory it can support in GiB. For each PM j, let $\mathbf{D}_j = \{G_j^1, G_j^2, ..., G_j^{|\mathbf{D}_j|}\}$ be the set of available physical disks, where $|\mathbf{D}_j|$ is the number of disks. For each of the physical disks, G_j^k represents the available disk volume size in GB. The capacity of PM j can be represented by $R_j = \{\mathbf{C}_j, B_j, \mathbf{D}_j\}$. Though we used three resources as an example here, our method can be easily extended to any number of resources by simply adding more resource dimensions.

For each $V_i \in \mathcal{V}$ and each $P_j \in \mathcal{P}$, let x_{ij} be the binary assignment variable, which is set to 1 if VM V_i is assigned to PM P_j and 0 otherwise. The binary variables y_{ikjl} are used for CPU assignment. y_{ikjl} is set to 1 if VM V_i is assigned to PM P_j and V_i's requested vCPU k is assigned to the physical CPU core l of PM j; it is set to 0 otherwise. Similarly, the binary variables z_{ikjl} are used for disk assignment. z_{ikjl} is set to 1 if VM V_i is assigned to PM P_j and V_i's requested virtual disk k is assigned to the physical disk l of PM j; it is set to 0 otherwise.

Constraints. The following constraints are required:

$$\Sigma_{j\in\mathcal{P}} x_{ij} = 1, \ i \in \mathcal{V} \tag{1}$$

$$y_{ikjl} \leq x_{ij}, \ i \in \mathcal{V}, j \in \mathcal{P}, k \in \mathbf{c}_i, l \in \mathbf{C}_j \tag{2}$$

$$\Sigma_{j\in\mathcal{P}} \Sigma_{l\in\mathbf{C}_j} y_{ikjl} = 1, \ i \in \mathcal{V}, k \in \mathbf{c}_i \tag{3}$$

$$\Sigma_{k\in\mathbf{c}_i} y_{ikjl} \leq 1, \ i \in \mathcal{V}, j \in \mathcal{P}, l \in \mathbf{C}_j \tag{4}$$

$$\Sigma_{i\in\mathcal{V}} \Sigma_{k\in\mathbf{c}_i} \alpha_i^k y_{ikjl} \leq A_j^l \tag{5}$$

$$\Sigma_{i\in\mathcal{V}} \beta_i x_{ij} \leq B_j, \ j \in \mathcal{P} \tag{6}$$

$$z_{ikjl} \leq x_{ij}, \ i \in \mathcal{V}, j \in \mathcal{P}, k \in \mathbf{d}_i, l \in D_j \tag{7}$$

$$\Sigma_{j\in\mathcal{P}} \Sigma_{l\in D_j} z_{ikjl} = 1, \ i \in \mathcal{V}, k \in \mathbf{d}_i \tag{8}$$

$$\Sigma_{k\in\mathbf{d}_i} z_{ikjl} \leq 1, \ i \in \mathcal{V}, j \in \mathcal{P}, l \in D_j \tag{9}$$

$$\Sigma_{i\in\mathcal{V}} \Sigma_{k\in\mathbf{d}_i} \gamma_i^k y_{ikjl} \leq G_j^l \tag{10}$$

Equ.(1) ensures that every VM is assigned to exactly one PM. Equ.(2)-(5) are the constraints on CPU assignment. Equ.(2) ensures that the requested vCPUs for VM i may be assigned to the physical cores of PM j only if VM i is assigned to PM j. Equ.(3) ensures that every requested vCPU must be assigned to exactly one physical core. Equ.(4) ensures that VM i cannot place more than one of its vCPUs to the same physical core; Equ.(3) and Equ.(4) together enforce the CPU anti-collocation constraints. Equ.(5) is the CPU capacity constraint. Equ.(6) is the resource capacity constraints posed by the total memory size of each PM j. Equ.(7)-(10) are the constraints on disk assignment, which are analogous to Equ.(2)-(5). Equ.(7) ensures that the requested virtual disks for VM i may be assigned to the physical disks of PM j only if VM i is assigned to PM j. Equ.(8) ensures that every requested virtual disk must be assigned to exactly one physical disk. Equ.(9) ensures that VM i cannot have more than one of its virtual disks assigned to the same physical disk; Equ.(8) and Equ.(9) together enforce the disk anti-collocation constraints. Equ.(10) is the disk capacity constraint.

Objective. We assume that a fixed operation cost is incurred for a PM as long as the PM is used by some VMs (i.e., some VMs are assigned to the PM) [6]. Specifically, when a PM j is turned on to serve some VMs, there is a fixed cost s_j associated with running the PM; when the PM is off, there is zero cost. The operation cost may include the energy cost when a machine is running and typical maintenance cost. The optimization objective is to minimize the total operation cost (i.e., minimize the number of PMs used).

Let o_j be a 0-1 variable indicating whether PM j is used by some VMs. In normal situations, an assignment should be feasible so that no VMs are rejected; otherwise, the cloud provider usually increases its datacenter capacity. When no VMs are rejected, the total payment by the customers is fixed. In that case, the optimization objective is to minimize the total operation cost:

$$\min_{x,y,o} \Sigma_{j\in\mathcal{P}} s_j o_j \tag{11}$$

In order to solve the above formulated VM allocation problem, a general solution is to apply the branch and bound algorithm [22]. However, solving the MIP problems is fairly complex [37], [31]. The complexity of this algorithm is determined by the number of variables and the number of constraints, which can be counted easily. In practice, a cloud datacenter contains tens of thousands of VMs and PMs []. Compared to solving an MIP problem with a small number of variables, there is a dramatic increase in the complexity of VM placement problems. In particular, when a problem is sufficiently complex, the branch and bound algorithm becomes inefficient and impractical, especially in cloud where there is high demand on low latency of VM placement. As a result, a heuristic algorithm is needed to quickly solve the VM placement problem with anti-collocation constraints.

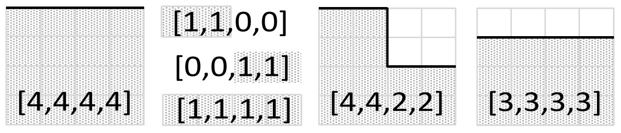
V. PageRank Based VM Placement

We develop a heuristic algorithm to solve the VM placement problem by ranking the PMs based on the probability that the PMs can fully utilize their resources across multiple resource dimensions by hosting VMs, and assigning a VM to the PM that has the highest rank by hosting such a VM. The rankings indicate how the PMs can fully utilized their resources in every resource dimensions.

A. Profile Ranking

For a given VM, to find a PM to host the VM among multiple PMs, we first calculate the profile of each PM after it hosts the VM. Based on the profiles, we will find the best PM to host the VM that will help fully utilize this PM's resources across different dimensions. In order to effectively and properly distinguish a preferred PM profile from the non-preferred ones, we propose to implement the PageRank algorithm [29] to rank the PM profiles. The best PM profile is the profile with the maximum value across all resource dimensions. For example, a profile $[4, 4, 4, 4]$ is regarded as the best profile of a PM with capacity $[4, 4, 4, 4]$. The PageRank based approach ranks the PM profiles according to their probabilities of transferring to the best profile. Considering the above examples, the PageRank based approach accurately reflects the quality of the profiles, which measures the tendency of fully utilizing PM resources. In the following, we will discuss how we design such a rank metric.

We consider one type of VMs as the VMs that have the same resource demand for each resource dimension. Given the best profile and the set of VM types, we define the *quality of a profile* as the capability of this profile to develop to the best profile by accommodating VMs from the VM set. For example, given a set of VM types $\{[1, 1], [1, 1, 1, 1]\}$ and a PM with capacity $[4, 4, 4, 4]$ as shown in Figure 2, we compare the qualities of two profiles $[4, 4, 2, 2]$ and $[3, 3, 3, 3]$. Figure 2 only shows a part of the permutations of VM resource demands as an example. In this case, profile $[3, 3, 3, 3]$ has higher quality than profile $[4, 4, 2, 2]$, because it is easier for $[3, 3, 3, 3]$ to develop to the best profile than $[4, 4, 2, 2]$. There are two ways for $[3, 3, 3, 3]$ to develop to the best profile: accommodating one $[1, 1, 1, 1]$ VM, and accommodating two $[1, 1]$ (i.e., $[1, 1, 0, 0]$ and $[0, 0, 1, 1]$) VMs. Accordingly, there is only one way for $[4, 4, 2, 2]$ to develop to the best profile: accommodating two $[1, 1]$ (i.e., $[0, 0, 1, 1]$) VMs. Profile $[3, 3, 3, 3]$ has higher quality than profile $[4, 4, 2, 2]$ and also satisfies the second requirement.



(a) Best profile (b) VM types (c) Profile 1 (d) Profile 2

Fig. 2. Example of preferred profiles.

It is necessary to point out that the ranking of the profiles is respected to the set of VM types. In other words, a change of the VM set can result in a different ranking. Look at the above example again, if the set of VM types is changed to $\{[1], [1, 1]\}$, profiles $[4, 4, 2, 2]$ and $[3, 3, 3, 3]$ have the same quality as they both have three ways to develop to the best profile: accommodating two $[1, 1]$ VMs; accommodating four $[1]$ VMs; accommodating one $[1, 1]$ VM and two $[1]$ VMs.

B. PageRank Algorithm

The PageRank algorithm was designed by Google to rank webpages in their search engine results. PageRank works by counting the number and weights of links pointing to a page to determine a rough estimate of how important the webpage is. The underlying assumption is that more important webpages are likely to receive more links from other webpages [29]. Specifically, it works by treating each backlink to a webpage as if it were a vote of support for that page, and by giving high weight to a vote if it comes from a webpage that itself has a high rank. In order to know the rank of the voting page to begin with, the PageRank algorithm must run iteratively until votes in all directions have had a chance to propagate.

We abstract the demanded resource of a VM across multiple dimensions as the demand resource profile of a VM, and abstract the used resource of a PM across multiple dimensions as the used resource profile of a PM. Instead of ranking webpages, our objective now is to rank each possible profile of a PM. A ranking of a VM placement decision is corresponding to the ranking of the PM usage resource profile after accommodating the VM. Specifically, we rank the PM usage profiles in terms of the probability of reaching the best profile, and make VM placement decisions based on the rank.

Suppose there are two PM usage profiles P_a and P_b. We consider P_a to be a vote of support for profile P_b if P_b can be transferred from P_a via hosting a VM from any types of the VMs. We define the procedure of forming a new PM usage profile as adding the requested VM resource demands to the PM usage profile. The vote for P_b is given a weight based on the rank of P_a itself. In order to know the rank of P_a to begin with, the algorithm must be run iteratively until votes in all directions have had a chance to propagate. In the general case, the PageRank value for any profile P_i can be expressed as:

$$PR(P_i) = \frac{1-d}{N} + d \sum_{P_j \in M(P_i)} \frac{PR(P_j)}{L(P_j)}, \tag{12}$$

where $P_1, P_2, ..., P_N$ are the profiles under consideration, $M(P_i)$ is the set of profiles that link to P_i, $L(P_j)$ is the number of outbound links of profile P_j, d is a damping factor, and N is the total number of profiles. We set the damping factor to 0.85 as generally assumed [8]. Once the rank of each profile has been computed, the VM placement problem is solved by placing the VM in the PM that gives the highest rank after allocating the VM to the PM.

Algorithm 1 shows the pseudocode for calculating the PageRank scores for each PM resource usage profile. The algorithm takes the PM multi-dimension resource capacity, the set of VM types $\mathcal{S}_v$, and a damping factor d as input. The algorithm first generates the profile graph G based on the PM capacity and VM types (Line 1). The profile graph is a graph with its nodes representing PM usage profiles, and its edges representing placing VMs of certain types to the profiles and producing new profiles. After the graph is generated, the algorithm defines two variables associated with each node (profile), $PR(P_i)$ and $Aux(P_i)$, where $PR(P_i)$ is the PageRank score of node P_i, $Aux(P_i)$ is an auxiliary variable that helps to calculate the PageRank scores of the profiles pointed to P_i (e.g., the second term in Equ. (12)). The algorithm initiates $PR(P_i)$ to $\frac{1}{N}$, and $Aux(P_i)$ to 0 (Lines 2-4). $Aux(P_i)$ is calculated based on the equation in Line 10. The algorithm then iteratively calculates the PageRank score for every node in the graph until the PageRank scores converge (Lines 6-18). For example, the PageRank scores are regarded as convergent when the difference between the PageRank score of every node and the PageRank score of this node calculated in the previous iteration is smaller than a threshold ϵ. Within each iteration, the algorithm checks each node in the graph. For each node $P_i \in G$ (Line 7), it finds out the nodes pointed by P_i (e.g., $S(P_i)$) (Line 8) and then updates the value of the auxiliary variable of every node pointed to by P_i based on their old auxiliary variable, the PageRank score of node P_i, and the number of nodes pointed to by P_i (e.g., $|S(P_i)|$) (Lines 9-11). The algorithm then iterates over each node, calculates the PageRank score of each node, and sets the auxiliary variable to 0 (Lines 13-16). The last step in each iteration is to normalize the PageRank scores (Line 17). Finally, for each profile, we compute its final PageRank value by multiplying the Best Possible Resource Utilization (BPRU) of this profile (Line 19). The BPRU of a profile is defined as the maximum resource utilization (which is within $[0, 1]$) that the profile can further reach by accommodating several other VMs, i.e., the maximum resource utilization among those of the endpoints of paths containing the profile. If a profile cannot accommodate any other VMs, then the BPRU of this profile is the resource utilization of itself. The purpose of this discounting step is to discount the rank of profiles that will not reach the best profiles finally, i.e., the endpoints of the paths containing such a profile are not the best profile. We refer to a profile's endpoints as the endpoints of the paths containing the profile. A lower BPRU value (i.e., a longer distance between the maximum resource utilization of a profile's endpoints and the best profile) leads to more discounting and vice versa.

As a result, the final rank of a profile represents the probability that this profile can reach the best profile or high resource utilization. Therefore, to choose a PM to host a given VM among multiple PMs, we calculate each PM's profile after hosing the VM and then choose the PM that has the highest rank, i.e., the PM that has the highest probability of reaching the best profile. For this purpose, we produce a Profile-PageRank score table from the graph, in which each profile is associated with a rank score.

The graph and Profile-PageRank score table are relatively stable during a certain period of time (e.g., one month). Only after many new VM types appear, the graph and table need to be updated. For example, Amazon predefines the types of VMs, so the graph and table do not need to update unless Amazon adds many more types of VMs.

ALGORITHM 1: Pseudocode for PageRank algorithm.

```
Input: PM with multiple dimension resource capacity;
       set of VM types $\mathcal{S}_v$, damping factor $d$;
Output: PageRank scores for every profile;
1  Generate profile graph $G$;
2  for each $P_i \in G$ do
3      $PR(P_i) = \frac{1}{N}$;
4      $Aux(P_i) = 0$;
5  end
6  while Pagerank not converging do
7      for each $P_i \in G$ do
8          $S(P_i) \leftarrow$the set of profiles pointed by $P_i$, which is derived by
           accommodating an additional VM from the set of VMs $\mathcal{S}_v$;
9          for each profile $P'_i \in S(P_i)$ do
10             $Aux(P'_i) = Aux(P'_i) + \frac{PR(P_i)}{|S(P_i)|}$;
11         end
12     end
13     for each $P_i \in G$ do
14         $PR(P_i) = \frac{1-d}{N} + d \times Aux(P_i)$;
15         $Aux(P_i) = 0$;
16     end
17     Normalize: $PR(P_i) = \frac{P_i}{\sum_j P_j}$;
18 end
19 $PR(P_i) = PR(P_i) * BPRU(P_i)$;
```

C. VM Allocation Algorithm

The goal of our initial VM allocation algorithm is to place all VMs in as few PMs as possible, ensuring that the aggregated demand of VMs placed in a PM does not exceed its capacity across each resource dimension. As the problem is analogous to the multi-dimension bin packing problem which is NP-hard, we propose a dimension-aware heuristic algorithm to solve this problem, which takes advantage of cross dimensional complimentary requirements for different resources based on the designed PageRank score.

Algorithm 2 shows the pseudocode for our initial VM allocation algorithm that allocates each VM in VM_list to the PMs in the system. This algorithm refers to the Profile-PageRank score table obtained from Algorithm 1 for finding a suitable PM for each VM in VM_list (Line 1). The mechanism keeps two lists for all PMs in the system: $used\_PM\_list$ includes all used PMs and $unused\_PM\_list$ includes all unused PMs. In order to minimize the number of PMs used, to find a PM to host a given VM, the mechanism first attempts to use a PM from $used\_PM\_list$, and then from $unused\_PM\_list$.

ALGORITHM 2: Pseudocode for initial VM allocation.

```
   Input: $max\_Score = 0$;
1  for each VM in $VM\_list$ do
2      for each PM in $used\_PM\_list$ do
3          if PM does not have sufficient resource then
4              Continue;
5          else
6              Derive the set of possible PM profiles after
               accommodating every permutation of the VM's profile;
7              Select the profile with the maximum PageRank score as
               the best profile of the VM on this PM;
8              if PageRank score $> max\_Score$ then
9                  $max\_Score \leftarrow$ PageRank score;
10                 $allocate\_PM \leftarrow$ assign the VM on this PM using
                   the best accommodation;
11             end
12
13     end
14     if $allocate\_PM \neq null$ then
15         return $allocate\_PM$;
16     else
17         for each PM in $unused\_PM\_list$ do
18             if PM has sufficient resource then
19                 $allocate\_PM \leftarrow$ this PM;
20                 Move the PM to $used\_PM\_list$; Exit;
21             else
22                 Exit; // no solution
23
24         end
25
26 end
```

First, the algorithm iterates the $used\_PM\_list$ to find the best PM (Lines 2-13). For each PM in $used\_PM\_list$, the algorithm checks whether the PM has sufficient resources to host the VM. If the PM does not have sufficient resources, the algorithm continues to check the next PM in $used\_PM\_list$ (Lines 3-4). If the PM has sufficient resources, the algorithm derives the PM resource usage profile after accommodating the VM. We need to consider the permutations of the VM's request. Specifically, the algorithm derives the set of possible usage profiles on the PM after adding every possible permutation of the VM's request, and then looks up the PageRank scores of all the possible usage profiles from the Profile-PageRank score table. We then select the profile with the maximum PageRank score as the best accommodation of the VM on this PM, and record the score of this profile (Lines 6-7). Finally, the algorithm finds the scores of all PM options. The algorithm then finds the maximum score and the corresponding PM. The PM that will result in the maximum score is chosen for the VM (Lines 9-10). If a suitable PM is found in the $used\_PM\_list$ (Line 14), then the algorithm return the PM (Line 15); otherwise, the algorithm continues to search the $unused\_PM\_list$ to find a suitable PM (Lines 17-24). For each PM in $unused\_PM\_list$, the algorithm also checks whether the PM has sufficient resources to host the VM (Line 18). If the PM does not have sufficient resources, the algorithm continues to check the next PM in $unused\_PM\_list$. On the other hand, if the PM has sufficient resources, the algorithm selects this PM for the VM, moves this PM to $used\_PM\_list$, and returns (Lines 19-20). If a suitable PM cannot be found in $unused\_PM\_list$, the algorithm returns with no solution (Line 22).

In summary, PageRankVM simply refers to the Profile-PageRank score table to determine the VM placement. The computation complexity is low compared to the previous methods that need to resolve a linear programming problem. Calculating the new profile of each PM in $used\_PM\_list$ after hosting the given VM may generate a certain overhead. To solve this problem, we can adopt the 2-choice method, in which two PMs are randomly selected and then the best one is selected. Based on the previous works on the 2-choice method, the 2-choice method will lead to exponential improvement over one choice, but a poll size larger than two gains much less substantial extra improvement [26], [4]. However, this is not the focus of this paper.

VI. Performance Evaluation

To evaluate the performance of PageRankVM, we have implemented the algorithm, and conducted a simulation study on CloudSim [9] and a GENI testbed experiment [16].

A. Experiment Setup

Simulation. We conducted experiments on CloudSim, a modern simulation framework for cloud computing environments. In the experiments, we allocate an increasing number of VMs, from 1000 to 3000 with an interval of 1000, to the PMs. We repeatedly carried out each experiment for 100 times and reported the results. At the beginning, the VMs are allocated to the PMs based on the corresponding algorithms. When a PM is overloaded, its VMs need to be migrated to other PMs. When the simulation is started, the simulator calculates the resource utilization status of all the PMs in the datacenter every 300 seconds, and records the number of VM migrations and the number of overloaded PMs (the occurrence of overloaded PMs) during that period. We simulate a period of 24 hours.

We follow the VM and PM setup in Amazon EC2 [14] as close as we can. We take a subset of the allowed VM types (classes) of Amazon's EC2. Their resource requirements are shown in Table I. We use a subset of the PM types in Amazon EC2, which are listed in the Table II. Cloud providers generally do not disclose the detailed capabilities of all their PMs. The amount of resources that each type of PMs is equipped with (shown in Table II) is largely our guess based on the information revealed on Amazon's web site. Given the diversity of

TABLE I
DESCRIPTION OF VM TYPES

VM Types	Virtual cores		Memory (GiB)	Virtual Disk	
	#	Speed (GHz)		#	Size (GB)
m3.medium	1	0.6	3.75	1	4
m3.large	2	0.6	7.5	1	32
m3.xlarge	4	0.6	15	2	40
m3.2xlarge	8	0.6	30	2	80
c3.large	2	0.7	3.75	2	16
c3.xlarge	4	0.7	7.5	2	40

physical hardware that vendors offer, the amount of resources listed in Table II can also be understood as a plausible sample.

GENI testbed. In this real GENI testbed experiment, since we are not allowed to virtualize requested machines, we used the VM instances to emulate the PMs and ran jobs on the instances to emulate the VMs. In other words, we emulate the VM placement process as assigning jobs to run on each instance. When a PM (instance) is overloaded, we need to select some VMs (jobs) running on the PM, kill the VMs (jobs) and continue them on the destination PMs. We configured a testbed on GENI with 10 VM instances, each of which has 4 CPU cores. The instances are connected to a switch via 1Gbps links. We use another instance to act as a centralized controller (also connected to the switch via a 1Gbps link), which is responsible for running the VM placement algorithms to assign the jobs on different instances. Later on in this paper, we will refer to the VM instances as PMs and the jobs as VMs.

When the experiment starts, the centralized controller calculates the resource utilization status of all the PMs in the datacenter every 10 seconds, and records the number of VM migrations and the number of overloaded PMs (the occurrence of overloaded PMs) during that period. We ran the GENI testbed experiment in a period of 4 hours.

We considered the CPU resource allocation only. To consider the anti-collocation constraints, the 4 physical CPU cores of the PM are regarded as a 4-dimensional vector. The VM types we used are [1,1], [1,1,1,1], which represent that the VMs demand 2 vCPUs from two CPU cores, and 4 vCPUs from four CPU cores, respectively. We assume that each physical CPU core can host 4 vCPUs in this experiment.

Workload. For the simulation, we used the workload trace [9] available in CloudSim to generate each VM's resource consumption. The trace contains the CPU utilization of each node in PlanetLab every 5 minutes for 24 hours. We randomly chose traces of the VMs in our experiments. We also carried out experiment using public available Google cluster usage trace [18], the trace represents 29 day's resource usage information from May 2011, on a cluster of about 11k machines. For the GENI testbed experiment, we only used the Google cluster usage trace [18]. The resource utilization trace from PlanetLab VMs and Google Cluster VMs are used to drive the VM resource utilizations in the simulation and the GENI testbed experiment.

Comparison Algorithms. In the experiment, we implemented PageRankVM. When a PM is overloaded in PageRankVM, for each VM on the PM, we check the PageRank value of the resulting profile of this PM after removing the VM. Then we select the VM that can result in the highest PageRank value to remove. The destination PM is then selected based on Algorithm 2.

TABLE II
DESCRIPTION OF PM TYPES

PM Types	Physical Cores		Memory (GiB)	Physical Disk	
	#	Speed (GHz)		#	Size (GB)
M3	8	2.6	64	4	250
C3	8	2.8	7.5	4	250

TABLE III
POWER CONSUMPTION VS. CPU UTILIZATIONS.

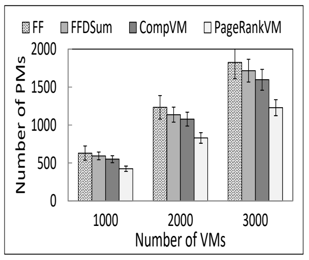
CPU util.	0%	20%	40%	60%	80%	100%
E5-2670 (W)	337.3	349.2	363.6	378	396	417.6
E5-2680 (W)	394.4	408.3	425.2	442.0	463.1	488.3

We compare PageRankVM with three algorithms, First Fit (FF) [27], First Fit Decreasing Sum (FFDSum) [30] and CompVM [10]. PageRankVM places VMs to PMs based on the PageRank scores of the resulting PM resource usage profile. FF places VMs to first PM that has sufficient resource to host this VM. FFDSum calculates the size of PM by weighted sum of the d-dimensional vector, which represents the resource capacity in d dimensions, and greedily places VMs to PMs with decreasing sizes. CompVM coordinates the requirements of resources and consolidates complementary VMs in the same PM. When a PM is overloaded in the above three methods, we use the default VM migration algorithm in CloudSim [9] to select VMs to migrate out. The destination PM for FF, FFDSum, and CompVM is then selected based on their own VM allocation algorithms, respectively. All algorithms use the strategy of PageRankVM to satisfy the anti-collocation constraints.

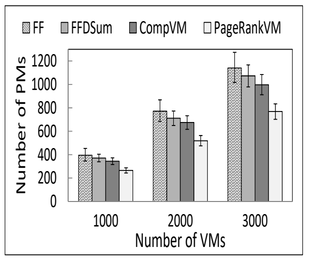
Energy Model. Energy consumption by PMs in datacenters is mostly determined by the CPU, memory, disk storage, power supplies and cooling systems [21], and the work in [6] gives the total energy consumption amount based on the CPU utilization. The configuration and power consumption characteristics of our used servers, M3 (High Frequency Intel Xeon E5-2670 v2 (Ivy Bridge) Processors [14]) and C3 (High Frequency Intel Xeon E5-2680 v2 (Ivy Bridge) Processors [14]), are scaled based on the CPU capacities and power characteristics of HP ProLiant ML110 G4 in [11], [23]. The power consumption characteristics of M3 and C3 are shown in Table III. Although these power consumption characteristics may not be exact accurate for the real processor models, it still sheds some light on the power consumption of the servers. Using this table, we calculate and compare the energy consumption of different algorithms. Since we leverage an estimation method to build the energy model and the model for the simulation and GENI testbed experiment is the same, we only evaluated the energy consumption in the simulation.

Comparison Metrics. We compared the VM placement algorithms in terms of the following metrics:

- *The number of PMs used.* It measures the resource efficiency of VM allocation algorithms to host all the VMs.
- *The energy consumption.* It measures the cumulated en-

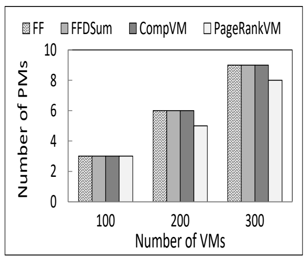


(a) PlanetLab trace.

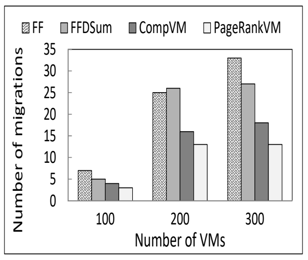


(b) Google Cluster trace.

Fig. 3. The number of PMs used in the simulation.

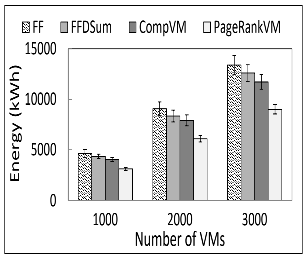


(a) The number of PMs.

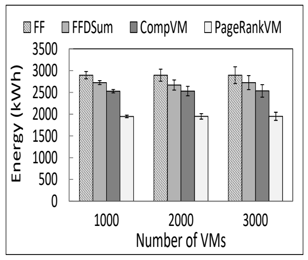


(b) The number of VM migrations.

Fig. 4. Results in the GENI testbed experiment.



(a) PlanetLab trace.



(b) Google Cluster trace.

Fig. 5. The energy consumption in the simulation.

ergy consumption of all PMs at the end of simulation.

- *The number of VM migrations.* It measures the number of VMs that are migrated in response to PM utilization overload during the experiment.
- *The SLO violations.* It is represented by the percentage of time, during which active hosts have experienced the CPU utilization of 100% [11], [35]. CPU utilization of 100% indicates that the cumulated CPU resource demands from all the VMs in a PM is the same as, or more likely has exceeded the CPU capacity of the PM.

B. The Number of PMs Used

Figures 3(a) and 3(b) show the median, the 1^{st} and 99^{th} percentiles of the total number of PMs used to provide service of the corresponding algorithms when the number of VMs was varied from 1000 to 3000 using the PlanetLab trace and the Google Cluster trace in the simulation, respectively. The result follows PageRankVM $<$ CompVM $<$ FFDSum $<$ FF. FF and FFDSum consider if there is sufficient resource in the PMs to host the VMs, while neglecting the resource utilization balance across different resource dimensions. This may result in some PMs that fully utilize one resource but under-utilize other resources. Especially, as we are considering the VM placement problem with anti-collocation requirement, previous single dimension resources such as CPU are now regarded multiple dimensions. This exaggerates the bad performance of the dimension unaware algorithms (e.g., FFDSum and FF). In contrast, CompVM outperforms FFDSum and FF, because it consolidates complementary VMs in different resource dimensions, thus fully utilizing each resource in each PM. PageRankVM needs fewer PMs than CompVM, because CompVM is a variation-aware approach, which is not as good as the PageRank based approach in terms of identifying a placement decision that has a higher potential to accommodate more VMs. The error bars show that the deviation of the results also follows the same order, which demonstrates the stability of PageRankVM.

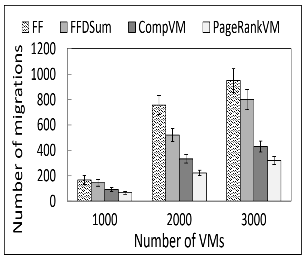
Figure 4(a) shows the total number of PMs needed versus different number of VMs in the GENI testbed experiment using the Google Cluster trace. We see that PageRankVM used fewer PMs than FF, FFDSum and CompVM when there are 200 and 300 VMs. However, the performance difference is not as large as the difference in the simulation. This is because the advantages of PageRankVM over other algorithms are more significant when the scale is relatively large and there are more resource dimensions. In this GENI testbed experiment, the number of PMs and the number of dimensions of resource vector are relatively small. Both simulation and GENI testbed results confirm the advantage of PageRankVM in reducing the number of PMs used.

C. Energy Consumption

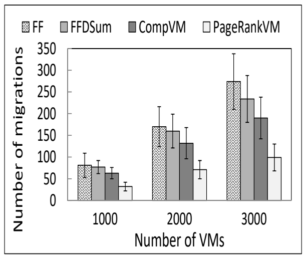
We then compare energy consumption of the datacenter during 24-hour simulations of the corresponding VM placement algorithms. The energy consumption is calculated according to the CPU utilization as shown in Table III. Figure 5(a) and Figure 5(b) show the median, the 1^{st} and 99^{th} percentiles of the energy consumption of the corresponding algorithms when the number of VMs was varied in the simulation. We see that the result follows PageRankVM$<$CompVM$<$FFDSum$<$FF. FF consumes highest energy, FFDSum consumes lower energy than FF, and CompVM consumes less energy than FFDSum. This is because FF has a high number of active PMs, while FFDSum has fewer than FF, and CompVM has fewer than FFDSum. PageRankVM outperforms the other algorithms in terms of energy consumption for two reasons. First, PageRankVM leads to a less number of active PMs than the other algorithms. Second, PageRankVM results in higher resource utilization on the active PMs. The range of the error bars also follows the same order, indicating the stability of PageRankVM in using fewer number of PMs to provide the same service than the other algorithms. These experimental results indicate that PageRankVM is more energy-efficient compared to other algorithms.

D. The Number of VM Migrations

VM migration is triggered when a PM becomes overloaded. In the experiment, an overloaded PM is flagged if the PM

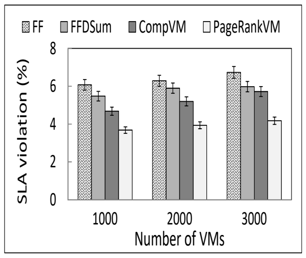


(a) PlanetLab trace.

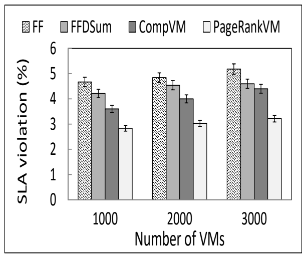


(b) Google Cluster trace.

Fig. 6. The number of migrations in the simulation.



(a) PlanetLab trace.



(b) Google Cluster trace.

Fig. 7. The SLO violations in the simulation.

resource utilization exceeds a threshold (i.e., 90%). Figure 6(a) and Figure 6(b) show the median, the 1^{st} and 99^{th} percentiles of the number of VM migrations of the corresponding algorithms when the number of VMs was varied in the simulation. We see that the result follows PageRankVM $<$ CompVM $<$ FFDSum $<$ FF. FFDSum and FF have high number of VM migrations because they fail to balance the PM resource utilizations across different dimensions. CompVM outperforms FFDSum and FF because it tries to balance the PM resource utilizations across multiple dimensions. PageRankVM generates fewer migrations than the other algorithms because it is able to avoid many VM migrations that are resulted from the overload of a single dimension. Also, PageRankVM can avoid overloading the destination PMs in the future by appropriately selecting PMs based on the PageRank scores of the resulting PM utilization profiles after accommodating the VM. The result of FFDSum and FF also indicate that they cannot reduce the number of VM migrations even though they use a higher number of PMs. The range of the error bars also follows the same order, due to the stability of PageRankVM in using fewer PMs as explained before.

Figure 4(b) shows the number of VM migrations versus different number of VMs in the GENI testbed experiment. Similarly, we see that PageRankVM leads to fewer VM migrations than FF, FFDSum and CompVM, due to the same reasons mentioned above.

E. SLO Violations

Figure 7(a) and Figure 7(b) show the median, the 1^{st} and 99^{th} percentiles of the SLO violations of the corresponding algorithms when the number of VMs was varied in the simulation. We see that the result follows PageRankVM $<$ CompVM $<$ FFDSum$<$FF. FF has the highest SLO violations, and CompVM is better than FFDSum and FF due to the reason that CompVM tries to balance the PM resource utilizations across multiple dimensions as explained before. PageRankVM has a lower percentage of SLO violations due to two reasons. First, PageRankVM considers different types of VMs, properly assigns multiple dimensional VM resource demands to PMs and hence avoids producing many overload PMs. Second, PageRankVM has a lower number of VM migrations as shown in Figure 6 which generates lower CPU consumption. The range of the error bars follows the same order, which confirms the stability of PageRankVM in constraining the number of SLA violations.

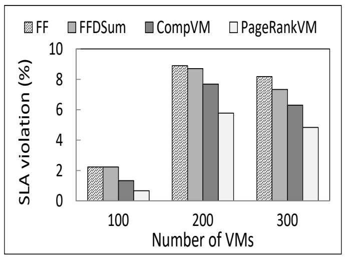


Fig. 8. The SLO violations in the GENI testbed experiment.

Figure 8 shows the SLO violations versus different number of VMs in the GENI testbed experiment. We see that PageRankVM results in fewer SLO violations than FF, FFDSum and CompVM, due to the same reasons explained above.

VII. Conclusions

In this paper, we studied the VM placement problem. Previous algorithms try to place VMs to PMs that can more fully utilize the resources and generate less variance in the resource usage across different dimensions. We show that these approaches do not necessarily improve the resource utilization in long term or reduce the number of PMs needed. We propose PageRankVM, a PageRank based VM placement algorithm with anti-collocation constraints. We abstract the multi-dimensional resource usages of a PM as a resource usage profile. PageRankVM ranks each possible resource usage profile of PMs after they host a given VM, in terms of the probability of fully utilize the resources across dimensions finally. PageRankVM also considers each physical hardware as one dimension and uses different permutations of a VM's resource request to satisfy the anti-collocation constraints in a lightweight manner. Compressive trace-driven simulation and GENI testbed experiments show the superior performance of PageRankVM in comparison with other heuristic algorithms. In the future, we will explore incorporating network infrastructure in designing PageRankVM in order to achieve bandwidth efficiency for the VM placement problem.

Acknowledgement

This research was supported in part by U.S. NSF grants OAC-1724845, ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751. We would like to thank Mr. Liuhua Chen for his help on this paper.

References

[1] Apache Cloud Stack, 2017.

[2] Y. Ajiro and A. Tanaka. Improving packing algorithms for server consolidation. In *Proc. of CMG*, pages 399–406.

[3] W. C. Arnold, D. Arroyo, W. Segmuller, M. Spreitzer, M. Steinder, and A. N. Tantawi. Workload orchestration and optimization for software defined environments. *IBM Journal of Research and Development*, 58(2/3):11–1, 2014.

[4] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. In *Proc. of ACM STOC*, 1994.

[5] Y. Azar, I. R. Cohen, S. Kamara, and B. Shepherd. Tight bounds for online vector bin packing. In *Proc. of the forty-fifth annual ACM symposium on Theory of computing*, pages 961–970, 2013.

[6] A. Beloglazov and R. Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[7] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera. A stable network-aware VM placement for cloud systems. In *Proc. of CCGRID*, pages 498–506, 2012.

[8] S. Brin and L. Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks*, 56(18):3825–3833, 2012.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *SPE*, 2011.

[10] L. Chen and H. Shen. Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters. In *Proc. of INFOCOM*, 2014.

[11] L. Chen, H. Shen, and K. Sapra. Distributed autonomous virtual resource management in datacenters using finite-markov decision process. In *Proc. of SOCC*, pages 1–13, 2014.

[12] L. Chen, H. Shen, and S. Sapra. RIAL: Resource intensity aware load balancing in clouds. In *Proc. of INFOCOM*, 2014.

[13] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira. Effective vm sizing in virtualized data centers. In *Proc. of IM*, pages 594–601, 2011.

[14] EC2 Instance Types, 2017. https://aws.amazon.com/ec2/instance-types/.

[15] Eucalyptus Systems, 2017. http://www.eucalyptus.com/.

[16] GENI Infrastucture, 2017. http://www.geni.net/.

[17] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[18] Google Cluster Data, 2017. https://code.google.com/p/googleclusterdata/.

[19] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *Proc. of CCR*, volume 44, 2014.

[20] Gurobi, 2017. http://www.gurobi.com/.

[21] M. Lauri and E. Brad. *Energy Efficiency for Information Technology: How to Reduce Power Consumption in Servers and Data Centers*. Intel Press, 2009.

[22] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[23] Z. Li, H. Shen, W. Ligon, and J. Denton. An exploration of designing a hybrid scale-up/out hadoop architecture based on performance measurements. *IEEE Transactions on Parallel and Distributed Systems*, 28(2):386–400, 2017.

[24] C.-C. Lin, P. Liu, and J.-J. Wu. Energy-aware virtual machine dynamic provision and scheduling for cloud computing. In *Proc. of Cloud*, 2011.

[25] L. Lu, H. Zhang, E. Smirni, G. Jiang, and K. Yoshihira. Predictive vm consolidation on multiple resources: Beyond load balancing. In *Proc. of IWQoS*, 2013.

[26] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proc. of ACM SPAA*, 1997.

[27] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proc. of CCGRID'09*, pages 124–131, 2009.

[28] OpenStack Project, 2017. http://www.openstack.org/.

[29] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[30] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder. Heuristics for vector bin packing. *MSR*, 2011.

[31] C. Qiu, A. Sarker, and H. Shen. Power distribution scheduling for electric vehicles in wireless power transfer systems. In *Proc. of SECON*, 2017.

[32] H. Shen and Z. Li. New bandwidth sharing and pricing policies to achieve a win-win situation for cloud provider and tenants. In *Proc. of INFOCOM*, 2014.

[33] H. Shen, L. Yu, L. Chen, and Z. Li. Goodbye to fixed bandwidth reservation: Job scheduling with elastic bandwidth reservation in clouds. In *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*, pages 1–8. IEEE, 2016.

[34] A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proc. of ACM/IFIP/USENIX Middleware*, pages 243–264, 2008.

[35] H. Wang and H. Shen. Proactive incast congestion control in a datacenter serving web applications. In *Proc.of INFOCOM*, 2018.

[36] Y. Xia, M. Tsugawa, J. A. Fortes, and S. Chen. Toward hierarchical mixed integer programming for pack-to-swad placement in datacenters. In *Proc. of ICAC*, pages 219–222, 2015.

[37] L. Yan, H. Shen, Z. Li, A. Sarker, J. A. Stankovic, C. Qiu, J. Zhao, and C. Xu. Employing opportunistic charging for electric taxicabs to reduce idle time. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(1):47, 2018.