Dynamic Tuning for Parameter-based Virtual Machine Placement

Abdelkhalik Mosa and Rizos Sakellariou
School of Computer Science
The University of Manchester
Manchester, UK
{abdelkhalik.mosa,rizos}@manchester.ac.uk

***Abstract*—Virtual machine (VM) placement is the process that allocates virtual machines onto physical machines (PMs) in cloud data centers. Reservation-based VM placement allocates VMs to PMs according to a (statically) reserved VM size regardless of the actual workload. If, at some point in time, a VM is making use of only a fraction of its reservation this leads to PM underutilization, which wastes energy and, at a grand scale, it may result in financial and environmental costs. In contrast, demand-based VM placement consolidates VMs based on the actual workload's demand. This may lead to better utilization, but it may incur a higher number of Service Level Agreement Violations (SLAVs) resulting from overloaded PMs and/or VM migrations from one PM to another as a result of workload fluctuations. To control the tradeoff between utilization and the number of SLAVs, parameter-based VM placement can allow a provider, through a single parameter, to explore the whole space of VM placement options that range from demand-based to reservation-based. The idea investigated by this paper is to adjust this parameter continuously at run-time in a way that a provider can maintain the number of SLAVs below a certain (predetermined) threshold while using the smallest possible number of PMs for VM placement. Two dynamic algorithms to select a value of this parameter on-the-fly are proposed. Experiments conducted using CloudSim evaluate the performance of the two algorithms using one synthetic and one real workload.**

***Index Terms*—Cloud computing; resource utilization; virtual machine placement; virtual machine consolidation**

I. Introduction

Cloud data centers house Physical Machines (PMs) or servers that host Virtual Machines (VMs) and associated components. Appropriate cloud management software (*e.g.* OpenStack [1], OpenNebula [2] or Eucalyptus [3]) can manage a large pool of resources, which, in addition to PMs, may include storage or networking resources. One of the tasks of the cloud management software is to assign VMs to PMs, a process which is known as *VM Placement* [4]. For example, the NOVA component of OpenStack uses the NOVA Filter Scheduler [5] to allocate VMs to PMs. The Nova-scheduler receives a VM request and determines where (that is, on what PM) it should run.

Mapping VMs to PMs statically does not fit well with the dynamic nature of large cloud data centers. As an illustration, the PMs in a cloud data center may be exposed to several changes over time. These may include deallocation of existing VMs, allocation of new VMs, fluctuations in the workload of the individual VMs, and so on. Also, some VMs may remain idle for prolonged time periods. In such a dynamic environment, mapping VMs to PMs statically may lead to PM underutilization; in fact, this is one of the leading causes of energy waste in cloud data centers [6]. Trying to increase PM utilization, some form of dynamic reallocation of the VMs becomes necessary to adapt to the diverse changes in cloud data centers and reduce overall energy consumption.

The dynamic reallocation of VMs can be either *reservation-based* or *demand-based* [7]. In reservation-based placement, VMs are assigned to PMs based on the requested (by the user) VM size (CPU, memory and bandwidth capacities) regardless of the actual utilization. Reservation-based placement may react to VM deallocation or new VM allocations by migrating VMs between PMs. Nevertheless, it does not respond to fluctuations of the workload, which means that VM resources may be reserved regardless of actual use. In contrast, VM reallocation based on the actual workload demand, rather than the reserved VM size, may potentially improve a PM's utilization. The improvement in utilization can be achieved by migrating VMs from underutilized (according to actual workload utilization) PMs. This migration process may help switch some PMs into one of the power saving modes whenever possible. However, such a migration, and demand-based placement in general, may result in Service Level Agreement Violations (SLAVs). A Service Level Agreement (SLA) defines the Quality of Service (QoS) requirements (*e.g.* availability of VMs) in the form of a contractual document between the service provider and the customer [8], [9]. An SLA violation (SLAV) occurs when previously agreed upon SLA requirements are not met. A common source of SLAVs is a result of VM migration due to the aggressive consolidation of VMs. This means that there is a trade-off between the utilization of PMs and SLAVs where better PM utilization may lead to a high number of SLAVs, whereas a low number of SLAVs may imply worse PM utilization.

In previous work [10], parameter-based placement has been proposed as a strategy that can generate a number of alternative allocations (beyond demand-based and reservation-based allocation) by means of setting the value of a single static parameter, α, which becomes an input to the VM placement problem. At the two ends of the range of values that this parameter α may take, 0 corresponds to demand-based allocation and 1 corresponds to reservation-based allocation. In principle,

the former will increase the number of SLAVs but should lead to better utilization than the latter. As the workload fluctuates over time, the main question that arises, from the cloud provider's point of view, is how to optimize utilization (and minimize energy consumption) without exceeding a certain number of SLAVs.

The question formulated above translates to the problem of dynamically calculating the values of this static parameter, α, in parameter-based allocation. This implies that the values of α should be chosen on the fly in such a way that the overall number of SLAVs does not exceed a certain threshold. This problem is addressed in this paper where two algorithms are proposed to allow a cloud provider to maintain the number of SLAVs within a certain threshold, at the same time optimizing utilization (also minimizing energy consumption), by dynamically choosing appropriate values of the parameter α on the fly.

The remainder of this paper is structured as follows: Section II reviews related work and Section III describes the problem and its characteristics. Then, Section IV presents the parameter-based VM placement and two dynamic approaches for calculating a value for the parameter α. Section V evaluates the proposed algorithms and Section VI concludes the paper and suggests some possible future directions.

II. Related Work

For recent surveys of the virtual machine allocation problem in cloud environments we refer to [4], [11]. In brief, the VM placement decision may be either static or dynamic. In static VM placement, the VM placement decision is made once during the initial placement of the VMs and the VM-to-PM mapping is never changed during the whole lifetime of the VM. In dynamic VM placement, the VM-to-PM mapping is usually reassessed periodically.

In reservation-based placement, heuristics such as first fit decreasing (FFD) or best fit decreasing (BFD), may be typically used to find a good static VM-to-PM mapping, as in [12] or [13], respectively. The goal of this mapping may be to minimize energy consumption (*e.g.* [13], [14]) or balance the load while considering communication costs (*e.g.* [15]). Wolke *et al.* [7] have analyzed simple bin packing algorithms for the initial placement of the VMs.

In demand-based VM placement, the decisions to change the VM-to-PM mapping are based on the changes in the cloud data center. Demand-based VM placement algorithms may aim to minimize energy consumption (*e.g.* [13], [16]–[19]) while the reallocation may be power-aware as in [20]. A dynamic VM placement controller may consider different management objectives, such as energy efficiency, load balancing, fair allocation, or service differentiation (*e.g.* [21]). The decision making policy for the dynamic reallocation of the VMs may be based on heuristics, as in [13], [22], [23], or it could be based on a utility function, as in [17], [24]. Dynamic VM placement solutions may incur a high number of SLAVs (in case of demand-based placement), or they may result in high energy waste (in case of reservation-based placement).

Parameter-based placement [10] explores VM placement solutions that can be perceived as solutions in the search space between demand-based and reservation-based placement. It gives the cloud provider additional flexibility to find a better trade-off between energy consumption (hence utilization) and the number of SLAVs. However, the parameter-based solution proposed in [10] relies on a static choice for the parameter α. As the workload fluctuates, one will need to recalculate appropriate values of α at each scheduling interval (meaning every time the VM-to-PM mapping is reassessed) to use as an input to the VM placement solution. This paper contributes two dynamic algorithms that can estimate the value of α for parameter-based VM placement on the fly.

III. Background and Problem Definition

A cloud provider offers VMs of different sizes, and the cloud management software maps VMs to PMs in a way that can save energy consumption while minimizing SLAVs. Two types of controllers may be used for VM placement: initial or dynamic VM placement controllers.

An initial VM placement controller receives VM placement requests and maps VMs to PMs based on the reserved VM size; this is equivalent to reservation-based placement and graphically highlighted in Fig. 1. Then, at each scheduling interval (in our evaluation, later in this paper, this is chosen to be every five minutes), the dynamic VM placement controller tries to optimize the current allocation based on the cloud provider's objectives.

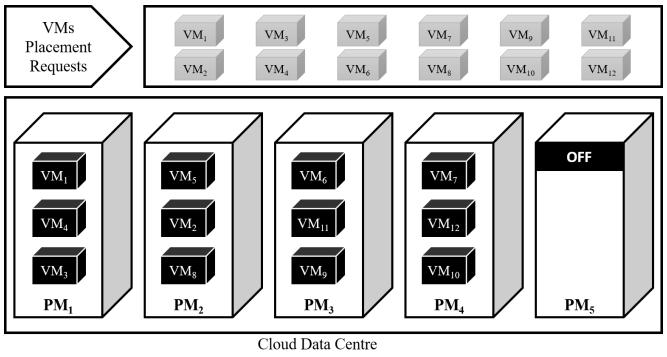

Fig. 1. Initial VM Placement

The dynamic VM placement controller reallocates VMs to PMs in the event of one of these two conditions:

1) Detection of an overutilized PM: Aggressive consolidation of VMs leads to overload situations for PMs. SLA violations due to PM overutilization (SLAVO) represent the time during which a PM is overutilized. A PM is overutilized when the PM utilization is higher than the maximum utilization threshold. SLAVO happens when the resource demand (this is bounded by a VM's capacity) of all hosted VMs is greater than the allocated resources. Equation 1 describes how to calculate SLAVO for PM i:

$$SLAVO_i = \frac{t_{oi}}{t_{ai}}, \tag{1}$$

where t_{oi} is the time period during which PM i is overutilized for one of its resource types (CPU, memory or bandwidth) and t_{ai} is the total time during which PM i is active (running).
Equation 2 computes the average SLAVO value for all PMs in a data center.

$$SLAVO = \frac{1}{N}\sum_{i=1}^{N}\frac{t_{oi}}{t_{ai}} \tag{2}$$

The dynamic placement controller reacts to a PM's overload situation by migrating one or more VMs from the overloaded PM to any of the suitable active (running) PM(s); alternatively, it may switch inactive PMs on to migrate a VM to, as shown in Fig. 2.

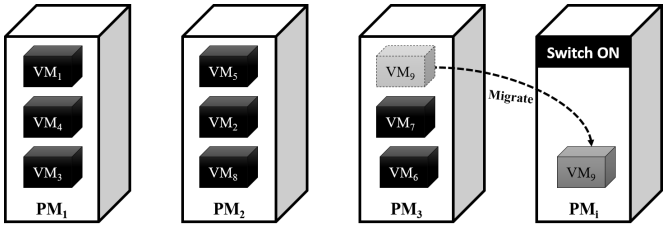

Fig. 2. Migrating a VM from an overutilized PM

2) Detection of an underutilized PM: A PM may be considered as *underutilized* when PM utilization is less than a predefined minimum utilization threshold. The dynamic placement controller reacts to an underutilized PM by migrating all VMs from the underutilized PM to other PM(s) whenever possible and switch that PM off or into a power saving mode, as shown in Fig. 3.

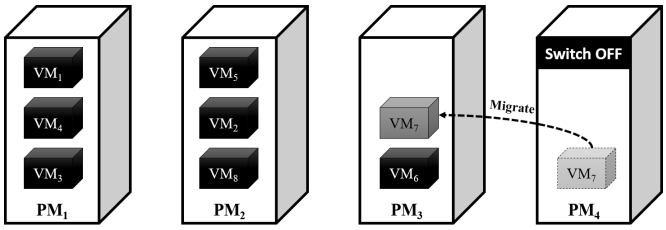

Fig. 3. Migrating VMs from underutilized PMs

The time required for the migration of a VM between two PMs depends on the available network bandwidth and the VM's memory utilization [25]. Equation 3 estimates the migration time of VM j supposing that the image and data of VMs are stored on a storage area network. In Equation 3, t_{mj} is the time required to migrate VM j; d_j^{ram} and a_j^{bw} represent VM j's memory demand and the available network bandwidth, respectively.

$$t_{mj} = \frac{d_j^{ram}}{a_j^{bw}}. \tag{3}$$

The migration of a VM, either because of underutilization or overutilization, results in SLA violations due to VM migration (SLAVM). Equation 4 estimates the SLAVM due to VM migration.

$$SLAVM = \frac{1}{m}\sum_{j=1}^{m}\frac{d_{dj}^{cpu}}{d_j^{cpu}}, \tag{4}$$

where d_{dj}^{cpu} is the VM$_j$'s under-allocated CPU demand due to migration, d_j^{cpu} is the total CPU demand by VM$_j$ and m is the total number of VMs.

Once the dynamic VM placement controller detects that some VMs require migration, it will try to find suitable PMs for the VMs migrated from both overutilized and underutilized PMs. The dynamic VM placement controller suffers from utilization inefficiency or it incurs a high number of SLAVs when it adopts either reservation-based or demand-based placement, respectively. In contrast, a parameter-based VM placement controller may allocate VMs based on the actual demand plus an additional margin (slack). In Section IV, we explore such a parameter-based VM placement solution through two proposed algorithms that calculate the value of the parameter α on the fly.

IV. Dynamic Parameter-based VM Placement

Parameter-based VM placement considers vertical scaling (i.e., resizing the VM), by scaling a VM's allocated resources (CPU, memory, bandwidth) up or down. In this paper, we assume that there is a predefined (by the provider) SLAVs *threshold* that defines a quantity for the number of SLAVs that should not be exceeded. The proposed parameter-based VM placement algorithms scale up a VM's allocated resources (bounded by the VM's maximum size) when the number of SLA violations exceeds the SLAVs threshold. On the other hand, the proposed algorithms will scale down the allocated resources when the number of current SLA violations is lower than the SLAVs threshold. Calculating the rate of scaling up or down is an integral part of the problem (and our proposed algorithms). This calculation is used to adjust the value of the parameter α, which acts as an autonomic knob that attempts to fine tune the resources allocated to each VM.

Parameter-based VM placement explores the space between the resources required for the actual workload (demand-based allocation) and the resources required for a VM at full capacity (reservation-based allocation). Fig. 4 shows the space in which parameter-based placement operates compared to both reservation-based and demand-based placement approaches. Parameter-based allocation adds an extra margin (slack) to the resources required to meet actual demand to accommodate some later sudden increase of the workload. This margin can be calculated based on the current value of the parameter α, as shown in Equation 5, which specifies how much is going to be allocated to VMj at time t.

$$\mathrm{a}_{jt}^{r} = \alpha \cdot (c_j^r - d_{jt}^r) + d_{jt}^r, \tag{5}$$

where a_{jt}^{r} is the resources of type r (CPU, memory, bandwidth) that will be allocated to VM$_j$ at time t; c_j^r is the capacity of resource r per VM_j (VM's size according to the reservation) and $d_j^r t$ is the total demand of resource r at time

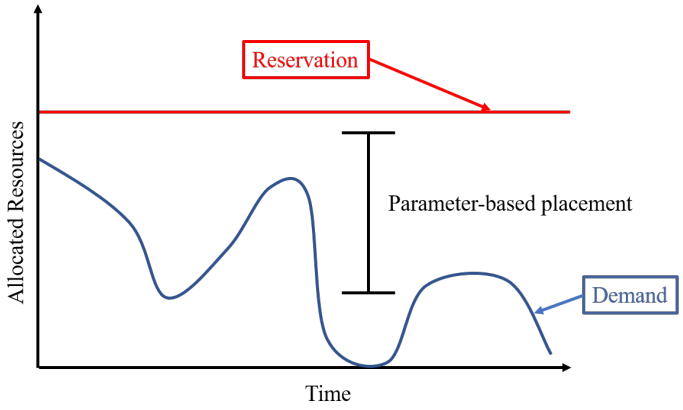

Fig. 4. Reservation vs demand vs parameter-based placement

t. Setting α to 0 will only allocate resources based on the current demand (in which case $a^r_{jt} < c^r_j$), whereas setting α to 1 means that the full VM size is allocated (in which case $a^r_{jt} = c^r_j$). This means that:

$$\alpha = \begin{cases} 0, & \text{demand-based placement} \\ 1, & \text{reservation-based placement} \end{cases}$$

For the parameter-based VM placement solution, we have used a modified version of the best fit decreasing algorithm (BFD) from [10]. This algorithm allocates VMs to PMs using the value of parameter α to check whether a PM is suitable to host a VM or not. At each scheduling interval, the parameter-based VM placement solution dynamically migrates selected VMs from the overutilized PMs as well as all VMs from the underutilized PMs. When an overloaded PM is detected, the placement algorithm will start by selecting VMs with the minimum memory utilization to migrate as this minimizes the overall migration time.

Adopting a parameter-based VM placement solution that keeps using the same value of α will not allow the cloud provider to set an SLAVs threshold that should not be surpassed. Such a statically predefined value of α cannot guarantee a number of SLAVs less than an SLAVs threshold (unless α is set to 1, or close to 1). This is because the number of SLAVs depends on both the current value of α (and hence the VM-to-PM placement) and the nature of the workload. Therefore, it becomes a challenge to compute the parameter α at each scheduling interval so that the resulting number of SLAVs does not exceed the SLAVs threshold (without underutilizing the resources). In what follows, two approaches that can dynamically estimate the value of α in two different ways, namely, *instantaneous* and *window-based*, are presented. They correspond to different methods to increase or decrease the value of α at each scheduling interval.

A. Instantaneous

The instantaneous approach estimates the value of the parameter α based on the current value of the number of SLAVs and its difference from the SLAVs threshold. The aim is to change promptly the rate of increase or decrease of the allocated resources (in this paper, we are only considering CPU resources) to adapt to workload fluctuations. Thus, we hope to reduce the number of SLAVs whenever this number exceeds the SLAVs threshold by increasing the value of α. Conversely, the value of α is decreased when the number of current SLAVs is lower than the SLAVs threshold. The changes in the value of α should drive efficient PM utilization without exceeding the predefined SLAVs threshold.

The instantaneous approach estimates a new value of α on the basis of the difference between the current number of SLAVs and the SLAVs threshold. To give a slight boost when this difference is close to zero, we take the square root of the difference; this seems to enable the algorithm to adapt promptly and makes it less sensitive to the initial value of the parameter α.

The instantaneous approach is summarized in Algorithm 1, which computes the value of the parameter α using the square root of the difference between the current number of SLAVs and the SLAVs threshold.

Algorithm 1 Estimating α using an instantaneous approach

```
1: currentSlav ← getCurrentSlav();
2: requiredSlav ← The non-exceeding SLAV threshold;
3: α ← get current value of α;
4: if (currentSlav ≥ requiredSlav) then
5:     sqrtRate ← √(currentSlav − requiredSlav);
6:     α = min(1, α+sqrtRate);
7: if (currentSlav < requiredSlav) then
8:     sqrtRate ← √(requiredSlav − currentSlav);
9:     α = max(0, α−sqrtRate);
10: return α;
```

B. Window-based

In the window-based approach, we make some use of the previous values for the number of SLAVs in addition to the current value (only the current value is used by the instantaneous approach). Thus, we look at a window of a specified size into the history of SLAVs to identify a trend (if there is one) with the preceding SLAVs (for example, whether the number of SLAVs is increasing or decreasing) to help estimate a more fine-grained value of α. As an illustration, in Fig. 5, the number of SLAVs at time$_{i+2}$ (SLAV$_{ti+2}$) is higher than the SLAVs threshold, and the number of SLAVs is increasing over Window 1, which is the area between time$_i$ and time$_{i+2}$. Consequently, we should increase the allocated resources by increasing the value of α at a high rate to reduce the number of SLAVs as soon as possible. Nevertheless, when the current number of SLAVs is lower than the SLAVs threshold and the number of SLAVs is increasing, as shown over Window 2 in Fig. 5, then we should decrease the value of the parameter α by a small rate to ensure less energy consumption through better utilization. The opposite happens when the number of SLAVs is decreasing over the specified window of the SLAVs history.

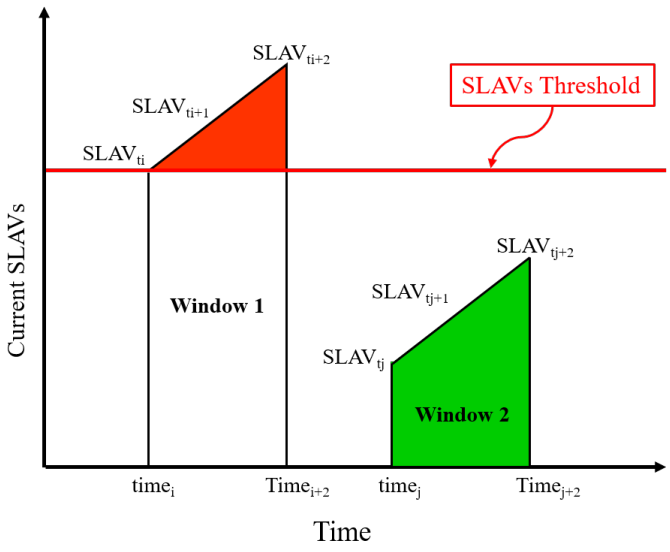

Fig. 5. Using a window of SLAVs history to estimate α

Algorithm 2 Estimating α using a window-based approach

```
1:  currentSlav ← getCurrentSlav();
2:  requiredSlav ← The non-exceeding SLAV threshold;
3:  α ← get current value of α;
4:  slavHistory[] ← stores previous SLAVs values;
5:  slavHistory.add(currentSlav);
6:  windowSize ← set window size;
7:  window[] ← selectedHistory(slavHistory, windowSize);
8:  if (currentSlav ≥ requiredSlav) then
9:      for (i = 0; i < len(window); i++ ) do
10:         if (window[i] > window[i+1]) then
11:             α = min(1, α+(currentSlav − requiredSlav));
12:             return α;
13:         else
14:             continue;
15:     α=min(1, α + √(currentSlav − requiredSlav ));
16: else
17:     for (i = 0; i< length(window); i++) do
18:         if (window[i] < window[i+1]) then
19:             α = max(0, α−(requiredSlav − currentSlav));
20:             return α;
21:         else
22:             continue;
23:     α=max(0, α − √(requiredSlav − currentSlav ));
24: return α;
```

Algorithm 2 describes the window-based approach that makes use of the previous values of SLAVs to calculate α. In Algorithm 2, we keep the history of SLAVs in the slavHistory array and save the selected history in the window array, which will include the previously stored values of SLAVs for the given window size. Then, we check whether the number of current SLAVs is greater or less than the SLAVs threshold. When the number of current SLAVs is greater than or equal to the SLAVs threshold and the number of SLAVs is not increasing, then we increase α by a small value as shown in Line 11. However, we increase α by a higher value when the number of SLAVs is increasing (in which case we use the square root, as shown in Line 15). On the other hand, when the number of the current SLAVs is less than the SLAVs threshold and the number of SLAVs is not decreasing, then we reduce α by a small value as shown in Line 19. Same as before, we decrease α by a higher value when the number of current SLAVs is less than the SLAVs threshold, and the number of SLAVs is decreasing as shown in Line 23.

V. Evaluation

A. Simulation Environment and Workload

We have implemented and evaluated the proposed algorithms using the widely used CloudSim simulation toolkit [26]. The simulated cloud data center has 800 PMs that run 1033 VMs. This setting is embedded in CloudSim and was chosen to correspond to the environment of real (PlanetLab) workload traces also used in our experiments. There are two types of PMs in the data center, namely, HP ProLiant ML110 (2 cores at 1860 MHz each) and HP ProLiant ML110 G5 (2 cores at 2660 MHz each) and can host four different VM types. For the detection of overutilized PMs, we have used a static overutilization threshold of CPU utilization; a PM is considered to be overutilized when CPU utilization is 100%. Whenever an overutilized PM is detected, we choose to migrate VMs with the minimum memory for migration from the overutilized PM to minimize the total migration time. The simulation works for a full day, and the dynamic reallocation of the VMs takes place every five minutes of simulated time (scheduling interval). Moreover, for the purposes of Algorithm 2, we have used a window of size two to capture the trend of previous SLAVs and check whether the number of SLAVs is increasing or decreasing.

To evaluate the proposed algorithms, we have used real workload traces from the PlanetLab platform [27] and synthetic workload traces. The PlanetLab workload traces represent the CPU utilization of the running VMs every five minutes (reallocation/scheduling interval). In the experiments, we have used the traces collected on 20 April 2011. For the synthetic workload, we randomly generate CPU utilization every five minutes based on a uniform random distribution with values in the range between zero and one.

B. Performance Metrics

We have used two performance metrics to measure the performance of the proposed approaches; namely, energy consumption and SLA violations (SLAVs). The energy consumption metric estimates the energy consumed for all running PMs in the cloud data center using CloudSim's SPECpower benchmark [28] to estimate energy consumption based on the actual CPU utilization. Table I, from [29], shows the energy consumption, in Watts, at different utilization levels.

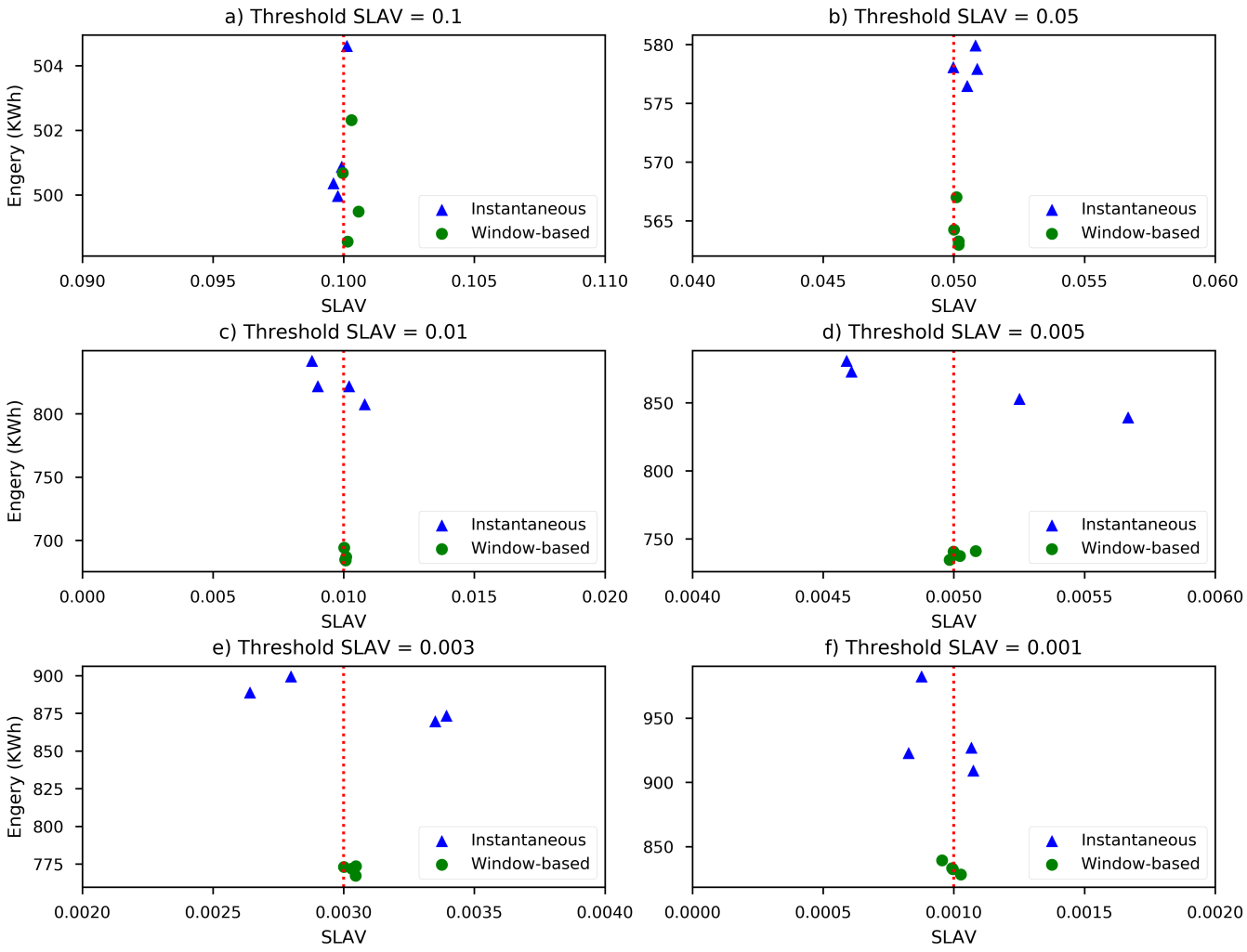

Fig. 6. Energy to SLAV using the synthetic workload

TABLE I
ENERGY CONSUMPTION (IN WATTS) AT DIFFERENT CPU UTILIZATION LEVELS [29]

Server	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP ProLiant G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP ProLiant G5	93.7	97	101	105	110	116	121	125	129	133	135

To calculate the number of SLA violations (SLAVs) in the data center, we used the SLAV metric, introduced in [29], which is a composite metric that considers two sources of violations. These two sources are violations resulting from overutilization of PMs (SLAVO) and violations resulting from the migrations of the VMs (SLAVM). Equation 6 describes how the number of SLAVs is calculated.

$$SLAV = SLAVO \times SLAVM \tag{6}$$

Here, $SLAVO$ corresponds to SLAVs from overutilization of PMs (computed according to Equation 2 in Section III), while $SLAVM$ estimates the performance degradation due to VM migrations (computed according to Equation 4 in Section III).

C. Experimental Results

We have conducted experiments to evaluate the estimation of α for the parameter-based VM placement solution using both the instantaneous approach and the window-based approach. For each of the two approaches, we have computed: (i) the amount of energy consumed by all PMs to allocate the VMs without exceeding the predefined SLAVs threshold, and (ii) the resulting value of SLAV. To run the two algorithms, we have to set an initial value for the parameter α in the algorithms. We have used four different initial values for α, from the list [0.2, 0.4, 0.6, 0.8], so that we can cover the range between zero (demand-based), and one (reservation-based), and test any sensitivity that each algorithm may have to the initial value of α.

For each type of workload, we have set six different SLAVs thresholds. For the synthetic workload, the chosen SLAVs thresholds are 0.1, 0.05, 0.01, 0.005, 0.003 and 0.001. We have started with 0.1 as the number of SLAVs resulting from

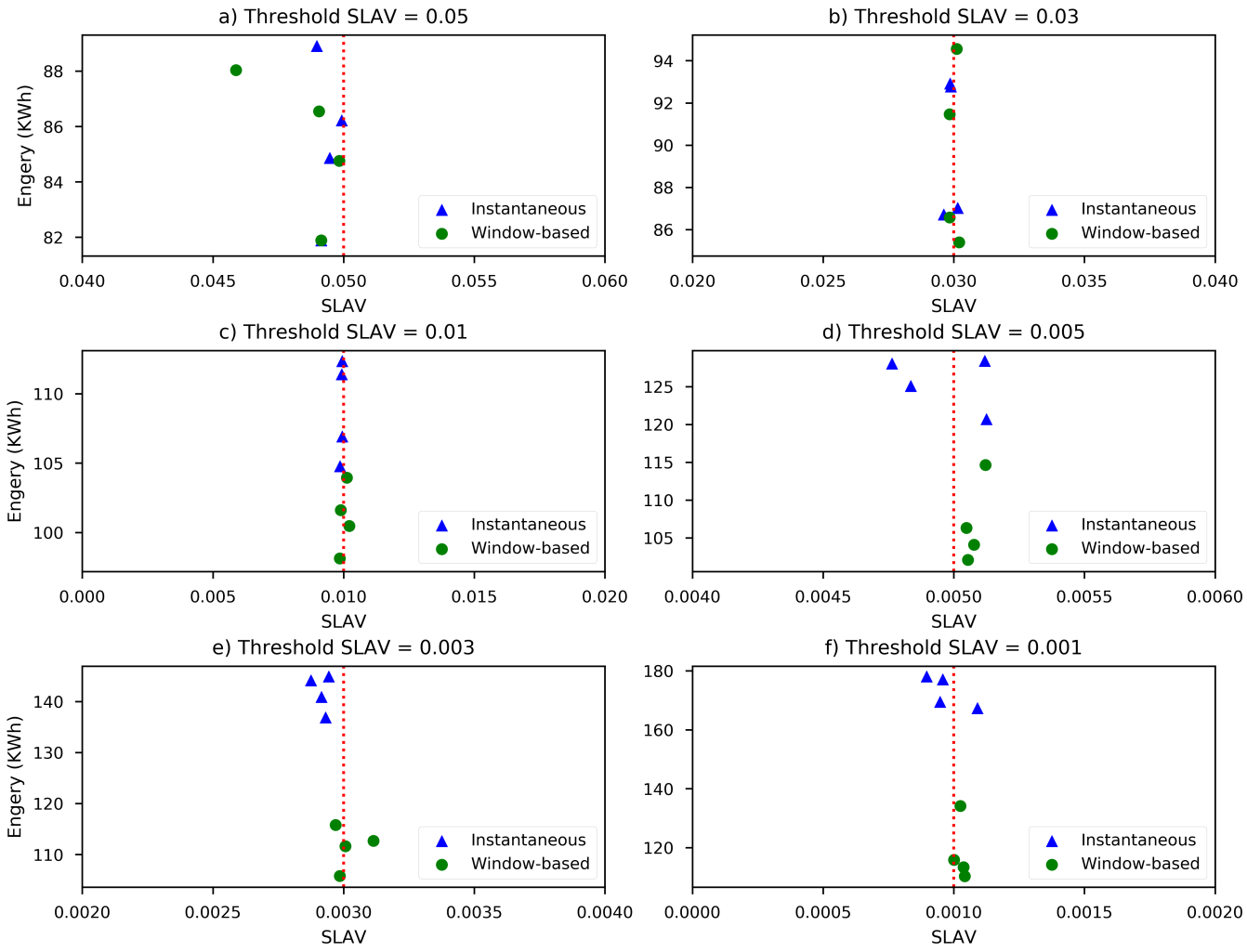

Fig. 7. Energy to SLAV using the PlanetLab traces

demand-based placement (i.e., $\alpha = 0$, when number of SLAVs is expected to be highest) is 0.1063. For the PlanetLab traces, the chosen SLAVs thresholds are 0.05, 0.03, 0.01, 0.005, 0.003 and 0.001. We have started with 0.05 as the number of SLAVs resulting from demand-based placement is 0.0542.

1) Experiment 1: Using a Synthetic Workload: Fig. 6 shows the amount of energy consumed and the resulting SLAVs based on the specified SLAVs threshold by running both the instantaneous and the window-based approaches using the synthetic workload. In each plot of Fig. 6, the X-axis represents the resulting SLAVs, and the Y-axis represents the amount of energy consumed to allocate the VMs to comply with the specified SLAVs threshold. The dotted vertical line represents the SLAVs threshold. Each point represents energy consumption to the number of SLAVs (cf. Equation 6) using a different initial value of α (initial α: 0.2, 0.4, 0.6 and 0.8). By and large, Fig. 6 demonstrates that the window-based approach performs better than the instantaneous approach as it can meet the SLAVs threshold while consuming less energy. However, for high values of SLAV thresholds, as in Fig. 6 (a), both approaches are somewhat sensitive to the initial value of α as shown in the spread of the points. Moreover, Fig. 6 (b), (c), (d), (e) and (f) show that the window-based approach becomes less sensitive to the initial value of α as the SLAVs threshold gets smaller, in contrast to the instantaneous approach.

2) Experiment 2: Using a PlanetLab Workload: This experiment aims to test the behaviour of both the instantaneous and window-based approaches using real workload traces from PlanetLab instead of the synthetic ones. Fig. 7 shows energy and SLAVs achieved by the two approaches. Fig. 7 still confirms that the window-based approach performs generally better than the instantaneous approach, resulting in less energy consumption for the specified SLAVs threshold. However, Fig. 7 demonstrates that the window-based approach generally becomes more sensitive to the initial value of α when the average utilization is very low (this may be because with PlanetLab traces the average workload is less than 13% of the maximum as opposed to an average of about 50% with

the synthetic workload). Furthermore, the results suggest that the window-based approach produces better results when the SLAVs threshold is set to smaller values. All in all, both experiments demonstrate that, as expected, the window-based approach can estimate the parameter α in a way that provides better results.

VI. Conclusion and Future Work

This paper investigates parameter-based VM placement, which aims to explore VM placement approaches that are in between the traditionally used demand-based and reservation-based placement. Two different approaches to estimate the value of the parameter α (used in parameter-based placement) on the fly, having as an objective not to exceed a certain SLAVs threshold, were proposed. One approach uses an instantaneous calculation to adapt the value of α, based on the distance between the number of current SLAVs and the SLAVs threshold, while another, window-based, approach makes use of recent history for the number of SLAVs. Comparing the two proposed algorithms, it appears that the window-based approach achieves better results than the instantaneous approach. This may be attributed to the more sophisticated strategy (and perhaps more conservative choices) used to select the rate with which to increase or decrease the value of α in the algorithm.

A list of possible directions for future work includes: (i) An investigation of different algorithms for estimating the value of the parameter α on the fly. For example, one can estimate the value of the parameter α based on a predicted behaviour for the workload. (ii) Further evaluation of the algorithms proposed on this paper using additional workload traces that might have different patterns.

Acknowledgment

The first author would like to acknowledge the support of the Egyptian Government during his PhD program.

References

[1] "OpenStack," https://www.openstack.org/, accessed: 2018-04-26.

[2] "OpenNebula," https://opennebula.org/, accessed: 2018-04-26.

[3] "Eucalyptus," https://github.com/eucalyptus/eucalyptus/wiki, accessed: 2018-04-26.

[4] I. Pietri and R. Sakellariou, "Mapping virtual machines onto physical machines in cloud computing: A survey," *ACM Computing Surveys (CSUR)*, vol. 49, no. 3, p. 49, 2016.

[5] "Openstack docs: Nova filter scheduler," https://docs.openstack.org/developer/nova/filter_scheduler.html, accessed: 2018-04-26.

[6] Nrdc.org, "America's Data Centers Consuming and Wasting Growing Amounts of Energy," 2015. [Online]. Available: http://www.nrdc.org/energy/data-center-efficiency-assessment.asp

[7] A. Wolke, B. Tsend-Ayush, C. Pfeiffer, and M. Bichler, "More than bin packing: Dynamic resource allocation strategies in cloud data centers," *Information Systems*, vol. 52, pp. 83–95, 2015.

[8] C. A. Lee and A. F. Sill, "A design space for dynamic service level agreements in openstack," *Journal of Cloud Computing*, vol. 3, 2014.

[9] M. Maurer, I. Brandic, and R. Sakellariou, "Enacting SLAs in clouds using rules," in *European Conference on Parallel Processing (Euro-Par)*. Springer, 2011, pp. 455–466.

[10] A. Mosa and R. Sakellariou, "Virtual machine consolidation for cloud data centers using parameter-based adaptive allocation," in *Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems*. ACM, 2017, p. 16.

[11] Z. Á. Mann, "Allocation of virtual machines in cloud data centers — a survey of problem models and optimization algorithms," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 11, 2015.

[12] L. Shi and B. Butler, "Provisioning of requests for virtual machine sets with placement constraints in IaaS clouds," *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pp. 499–505, 2013.

[13] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[14] N. Quang-Hung, P. D. Nien, N. H. Nam, N. H. Tuong, and N. Thoai, "A genetic algorithm for power-aware virtual machine allocation in private cloud," in *Information and Communication Technology-EurAsia Conference*. Springer, 2013, pp. 183–191.

[15] M. G. Rabbani, R. Pereira Esteves, M. Podlesny, G. Simon, L. Zambenedetti Granville, and R. Boutaba, "On tackling virtual data center embedding problem," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 177–184.

[16] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah, "Hybrid resource provisioning for minimizing data center SLA violations and power consumption," *Sustainable Computing: Informatics and Systems*, vol. 2, no. 2, pp. 91–104, 2012.

[17] A. Mosa and N. W. Paton, "Optimizing virtual machine placement for energy and sla in clouds using utility functions," *Journal of Cloud Computing*, vol. 5, no. 1, 2016.

[18] F. Farahnakian, A. Ashraf, T. Pahikkala, P. Liljeberg, J. Plosila, I. Porres, and H. Tenhunen, "Using ant colony system to consolidate vms for green cloud computing," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 187–198, 2015.

[19] D. More, S. Mehta, P. Pathak, L. Walase, and J. Abraham, "Achieving energy efficiency by optimal resource utilisation in cloud environment," in *Cloud Computing in Emerging Markets (CCEM), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1–8.

[20] J. V. Wang, C.-T. Cheng, and K. T. Chi, "A power and thermal-aware virtual machine allocation mechanism for cloud data centers," in *Communication Workshop (ICCW), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2850–2855.

[21] F. Wuhib, R. Yanggratoke, and R. Stadler, "Allocating Compute and Network Resources Under Management Objectives in Large-Scale Clouds," *Journal of Network and Systems Management*, pp. 1–26, 2013.

[22] L. Shi, B. Butler, D. Botvich, and B. Jennings, "Provisioning of requests for virtual machine sets with placement constraints in iaas clouds," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 499–505.

[23] M. A. Khoshkholghi, M. N. Derahman, A. Abdullah, S. Subramaniam, and M. Othman, "Energy-efficient algorithms for dynamic virtual machine consolidation in cloud data centers," *IEEE Access*, vol. 5, pp. 10 709–10 722, 2017.

[24] J. Simo and L. Veiga, "Partial utility-driven scheduling for flexible sla and pricing arbitration in clouds," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 467–480, 2016.

[25] W. Dargie, "Estimation of the cost of vm migration," in *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*. IEEE, 2014, pp. 1–8.

[26] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[27] "Planetlab workload traces," https://github.com/beloglazov/planetlab-workload-traces, accessed: 2018-04-26.

[28] "Specpower_ssj 2008," https://www.spec.org/power_ssj2008/, accessed: 2018-04-26.

[29] 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*, vol. 24, no. 13, pp. 1397–1420, 2012.