Virtual Machine Migration in Cloud Infrastructures: Problem Formalization and Policies Proposal

Alessandro Vittorio Papadopoulos, Martina Maggio

***Abstract*— Cloud computing has dramatically simplified the deployment of new software and, indeed, the number of applications that are hosted by cloud providers every day is increasing. The data center owner should provide computing capacity to a set of customers, each of them powering up and down virtual machines dynamically, to handle variations in the incoming requests. Cloud providers, however, should also optimize for quantities like energy consumption and managements costs, therefore trying to host all the customers virtual machines with the fewest amount of physical hardware machines possible. This leads to virtual machine co-location and potential performance inefficiency. To limit the inefficiency, virtual machines are migrated from one physical machine to another when overload conditions are detected. This paper analyzes the problem of virtual machine migration and presents some heuristic solutions to decide when to migrate a virtual machine from a physical machine to a different one. Experimental results show the differences between the proposed heuristics, providing a basis for a fair comparison among the techniques.**

Keywords: Cloud Computing, Virtual Machine Migration, Systems' Theory.

I. Introduction

The massive introduction of virtualization contributed to the success of computation paradigms like cloud computing. Data centers use virtualization and host a set of Virtual Machines (VMs) onto a set of Physical Machines (PMs) providing efficient resource utilization. Efficient resource utilization has been one of the main drivers during the data center evolution. Using a data center, one can reduce the hardware and operational costs [8] and also take into account energy consumption and environmental concerns [14], [17].

The entire idea of a data center is based on two main concepts: virtualization [2] and migration [7]. VMs can be moved from one PM to another, for example in order to turn off the source PM and save energy. In general, migration is a useful tool for administrators of data centers and clusters. A VM migration is transparent to the applications and modern virtualization technology invariantly support it [7], [21], [26]. It allows a clean separation between hardware and software, and facilitates fault management, load balancing, and low-level system maintenance. Moreover, it enables efficient resource utilization. Migration, however, comes at the cost of packaging the data for the VM to be migrated and sending it on the network from the source PM to the destination machine. This might be problematic if there are bandwidth limitations (both for LANs and WANs) in the data center [25] and can result in temporary unavailabilities due to migration latency [18].

Another known problem with VM migration comes from the necessary detection of workload hotspots, that should be the migration initiator. Due to this difficulty, initiating a migration was initially a manual task [26]. Manually-initiated migration lacked the necessary reactivity to respond to sudden workload changes and was error-prone, since each reshuffle might require migrations or swaps of multiple VMs to re-balance system load. For these reasons in the last years, industrial and academic researchers focused on how to improve and automatize the process of migrating VMs in data centers and clusters [10], [12], [15], [20].

The problem of VM migration is indeed a decision-making problem. Based on some feedback received from the data center, a policy should decide *when*, *how*, and *which* VMs have to be migrated, depending on different objectives — e.g., consolidating load onto fewer PMs, balancing the load on the active PMs, maintaining the quality of service promised to the users and much more. Splitting the problem in the three separate components, the *how* question has been effectively addressed with different live migration techniques [7], [18], [23], such as pre-copy [7], [11] and post-copy [13]. Based on the solution of this technological problem, and on the (realistic) assumption that the technology to perform live migration is available, it is possible to address the questions of *when* to migrate and *which* VM should be migrated. This is indeed a control problem, the technology provides an actuator (live migration) that can be used to satisfy the goals of the data center owner. There are many potential formulations of this problem, based on different modeling techniques like queuing network, equations, or Markov models. Also, there are many potential objectives, the most relevant two being energy savings and load balancing. Given these two remarks, the potential solutions for this control problem are many. This paper investigates some of these solutions for the load-balancing problem and compare the results obtained with them by means of simulating the execution of the migration manager in the same conditions, in a simulation environment.

The remaining of this paper is organized as follows. Section II describes some of the main techniques that are currently used to solve the VM migration problem. Section III provides a precise formalization of the problem and of the

A.V. Papadopoulos, Department of Automatic Control, Lund University, Sweden, alessandro.papadopoulos@control.lth.se

M. Maggio, Department of Automatic Control, Lund University, Sweden, martina.maggio@control.lth.se

This work was partially supported by the Swedish Research Council (VR) for the projects "Cloud Control" and "Power and temperature control for large-scale computing infrastructures", and through the LCCC Linnaeus and ELLIIT Excellence Centers.

proposed migration manager, together with some migration policies. Section IV presents the obtained simulation results, and Section V concludes the paper.

II. Related work

Many reasons motivate the use of migration in modern data center. One of the most prominent reasons is *server consolidation*, where the objective is to pack the load on the least possible amount of PMs that would still ensure some performance characteristics. Most of the existing migration strategies for server consolidation rely on eager migrations, that try to minimize the amount of PMs turned on. VMs that have been assigned the same PM end up sharing the resources according to different models [10] and having problems due to co-location [19].

In [9], the authors proposed a Linear Programming (LP) formulation and heuristics to control the VM migration. The approach prioritizes VMs with steady capacity, and aims at minimizing the number of PMs required to host a group of VMs. The server consolidation problem can be mapped to the multidimensional bin-packing problem [16], just by considering each VM as an item, the dimensions as the required capacities, and the goal is to minimize the PMs hosting the VMs, while respecting the physical capacities. This problem is NP-complete and was addressed both through LP and various heuristics. Another notable approach is based on an exponentially weighted moving average prediction model of the workload [27], and on the introduction of a quantity that is aimed at quantifying the unevenness in the utilization of multiple resources on a server, i.e., the *skewness*. Minimizing the skewness can improve the overall utilization of servers in the face of multidimensional resource constraints. In [3], the authors propose novel adaptive heuristics for dynamic consolidation of VMs based on an analysis of historical data from the resource usage by VMs. The algorithms proposed therein were proven to reduce energy consumption, while ensuring a high level of adherence to prescribed service level agreements.

All these solutions tend to co-locate many of the VMs by construction and to generate unnecessary migrations when VMs are subject to unpredictable workloads. Other techniques look at the number of PMs as a fixed given quantity and try to balance the load on the amount of machines that are turned on [1], [4]–[6], [22]. These are mainly heuristics that try to optimize different objectives, possibly conflicting [22]. These techniques may focus on a stream of deploy and undeploy requests [4], or be based on different tools, like Markov Decision Processes [5] or their extensions [6], and greedy strategies [1]. We advocate that different solutions should be compared, and therefore propose a comparison of some strategies to handle migrations.

III. Migration manager

A. Terminology

The problem addressed in this paper is the placement of a set $\mathbb{V} = \{v_1, v_2, \ldots, v_{n_v}\}$ of n_v VMs on a set $\mathbb{P} = \{p_1, p_2, \ldots, p_{n_p}\}$ of n_p PMs. In the following, $k \in \mathbb{N}$ represents the discrete time, and it counts the time instants when the migration manager can take a decision.

Definition 1 (Location function): The *location function* is a map $L : \mathbb{N} \times \mathbb{V} \times \mathbb{P} \to \{0,1\}$ that, for a given discrete time instant $k \in \mathbb{N}$, gives 1 if the VM $v \in \mathbb{V}$ is running on the PM $p \in \mathbb{P}$, 0 otherwise. The location function is such that

$$\sum_{p \in \mathbb{P}} L(k,v,p) = 1, \qquad \forall k \in \mathbb{N}, \forall v \in \mathbb{V} \tag{1}$$

$$0 \leq \sum_{v \in \mathbb{V}} L(k,v,p) \leq n_v, \qquad \forall k \in \mathbb{N}, \forall p \in \mathbb{P} \tag{2}$$

where (1) means that each VM *must* run on exactly one PM, and (2) indicates that a PM can be either empty (no VM is running on it), or can run different VMs.

Definition 2 (Virtual Machine): A *Virtual Machine* $v \in \mathbb{V}$ is a collection $v = (p_v, \ell_v^\circ, \ell_v, m_v^\circ, m_v, \mu_v, \pi_v)$, where

- $p_v \in \mathbb{P}$ is the PM on which the VM is running.
- $\ell_v^\circ \in \mathbb{R}$ is the *nominal CPU demand* of the VM when running on p_v.
- $\ell_v(\cdot) : \mathbb{N} \to \mathbb{R}$ is the *actual CPU demand* of the VM at time k; this quantity is time varying and depends on what is happening inside the VM. This quantity is bounded as

$$0 \leq \ell_v(k) \leq \ell_v^\circ, \qquad \forall k \in \mathbb{N}. \tag{3}$$

- $m_v^\circ \in \mathbb{R}$ is the *nominal memory* required by the VM (here expressed in GB).
- $m_v(\cdot) : \mathbb{N} \to \mathbb{R}$ is the *actual memory* occupied by the VM (here expressed in GB).
- $\mu_v(\cdot) : \mathbb{N} \to \mathbb{N}$ is the *number of migrations* that the VM has experienced, as a function of the discrete time.
- $\pi_v \in \Pi = \{\textsc{Gold}, \textsc{Silver}, \textsc{Bronze}, \textsc{Basic}\}$ is the service plan that the customer associated with the VM.

Without loss of generality we assume that ℓ_v°, m_v°, and π_v do not vary over time. In case a customer wants to upgrade one of the VMs he owns, adding some cores to it, increasing the required memory, or upgrade the service plan, a new machine is generated and the previous one is removed. Notice that the ideas presented in this paper apply also to the case when less or more service plans Π are defined.

A VM is here also characterized by the actual load $\ell_v(k)$ and memory $m_v(k)$ utilization. VMs are usually configured during creation with a specific amount of required resources, but at runtime the actual utilization is less than required one. This fact has been exploited in [24] with an *overbooking* approach, i.e., they allocate the VMs according to their actual resources request, with the assumption that the VMs' size will be usually less than the nominal one. In this paper we adopt the same idea.

Definition 3 (Nominal volume): Similarly to what has been done in [26], one can define the *nominal volume* of a VM $v \in \mathbb{V}$ as

$$V_v^\circ := \ell_v^\circ \cdot m_v^\circ. \tag{4}$$

Definition 4 (Actual volume): The *actual volume* of a VM $v \in \mathbb{V}$ is

$$V_v(k) := \ell_v(k) \cdot m_v(k). \tag{5}$$

Definitions 3 and 4 give a synthetic information on the resource utilization of a VM and are here used in the migration policies. Notice that the definitions of nominal and actual volumes can be easily generalized to the case where more resources are considered in the problem.

Definition 5 (Physical Machine): A *Physical Machine* $p \in \mathbb{P}$ is a collection $p = (c_p, f_p, M_p, \lambda_p)$, where

- $c_p \in \mathbb{R}$ is the number of cores associated with the PM.
- $f_p \in \mathbb{R}$ is the frequency of the cores of the PM.
- $M_p \in \mathbb{R}$ is the memory of the PM (here expressed in GB).
- $\lambda_p(\cdot) : \mathbb{N} \to \mathbb{R}$ is the *physical load*, i.e., the sum of the loads that all the VMs hosted on the PM demand at a given time. It can be defined as

$$\lambda_p(k) := \sum_{v \in \mathbb{V}} L(k, v, p) \ell_v(k). \tag{6}$$

For the sake of simplicity, and without loss of generality, in the rest of the paper we consider homogeneous PMs, i.e., all the cores with the same frequency. If this is not true, one can simply normalize the number of cores to the frequency of the slowest machine in the data center with a scaling factor

$$\sigma = \max_{p \in \mathbb{P}} f_p^{-1}.$$

Let us consider an example of non-homogeneous machines. Assume that we have two PMs, p_1 and p_2, and that p_1 has 16 cores running at $f_1 = 4.5\,\text{GHz}$ and p_2 has 8 cores running at $f_2 = 2.5\,\text{GHz}$. Then, the scaling factor is $\sigma = 1/2.5\text{GHz} = 0.4\text{ns}$. Then one obtain the equivalent number of cores for p_1 as $c_{p_1} = 16 \cdot f_1 \sigma = 28.8$; obviously, for p_2, we have that $c_{p_2} = 8 \cdot f_2 \sigma = 8$.

Sticking to the non-homogeneous machines example, the nominal CPU demand becomes a function of the PM on which v is running, i.e., $\ell_v^\circ = \ell_v^\circ(p_v)$. Thus, if the customer requires two cores, on p_2 the nominal CPU demand would be $\ell_v^\circ(p_2) = 2$, while on p_1 it would become $\ell_v^\circ(p_1) = 2/(f_1\sigma) = 1.11$.

Analogously to the VM case, one can define the *volume* of a PM $p \in \mathbb{P}$ as

$$W_p = c_p \sigma f_p M_p. \tag{7}$$

Definitions 6 and 7 directly follow from (7), and the quantities above.

Definition 6 (Nominal available volume): The *nominal available volume* NAW_p of $p \in \mathbb{P}$ is the nominal residual capacity in the physical machine, and it can be computed as

$$NAW_p(k) = W_p - \sum_{v \in \mathbb{V}} L(k, v, p) V_v^\circ \tag{8}$$

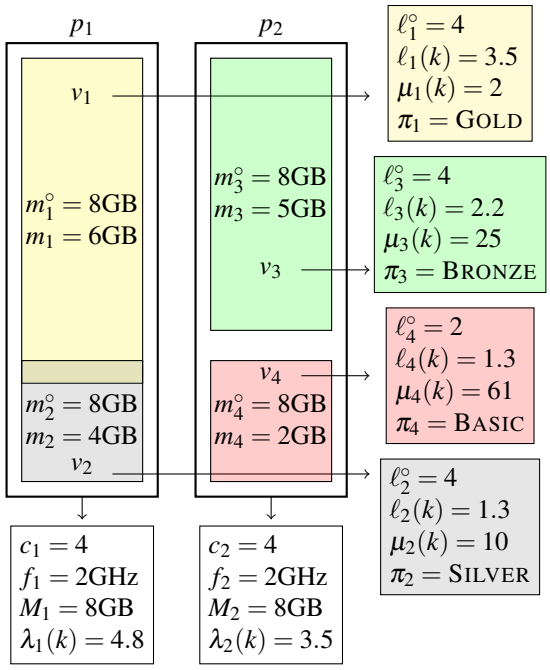

Fig. 1: Illustration of terminology.

Definition 7 (Actual available volume): The *actual available volume* AAW_p of $p \in \mathbb{P}$ is the actual residual capacity in the physical machine, and it can be computed as

$$AAW_p(k) = W_p - \sum_{v \in \mathbb{V}} L(k, v, p) V_v(k) \tag{9}$$

Figure 1 shows a graphical representation of the presented quantities at time k. In this case, there are 2 PMs, $\mathbb{P} = \{p_1, p_2\}$, hosting 4 VMs, $\mathbb{V} = \{v_1, v_2, v_3, v_4\}$. The first PM is loaded with a GOLD and a SILVER VMs, respectively v_1 and v_2. v_1 has a nominal CPU demand of 4 cores, but is using only 3.5 of them at time k, and has a nominal memory of 8GB, but using only 6GB at time k. Its nominal volume is $V_1^\circ = 32$, while its actual volume is $V_1(k) = 21$. It has also been migrated twice since the staring time. v_2 has a nominal demand of 4 cores but it is using only 1.3 of them at time k and a nominal memory of 8GB, but using only 4GB at time k. Its nominal volume is $V_2^\circ = 32$, and its actual volume is $V_2(k) = 5.2$. It has already been migrated 10 times. While p_1 has only 4 cores, the two machines are co-located, and interfering with each other. The volume of p_1 is $W_1 = 32$, and the total demand λ_1 is equal to 4.8. The nominal available volume is $NAW_{p_1} = -32$, and its actual available volume is $AAW_{p_1} = 5.9$. The two machines are therefore sharing some of the computing capacity and not obtaining the amount that they are requiring at time k, but the actual utilization of the VMs is still sufficiently low to be hosted by p_1. Similar considerations hold for the second PM p_2.

B. Load monitor

To identify and avoid overload situations – also called hotspots in [26] – one can define overload indexes for the PMs belonging to the data center. We propose to migrate

VMs based on the observation of overload indexes. For a given PM $p \in \mathbb{P}$, we define the integrated overload index $IOI_p(k)$. It represents the integral of the weighted fraction of load that exceeds a prescribed maximum capacity $\overline{\lambda}_p(k)$ on each PM. Denoting with x^+ the positive part of the number x, and with Δk the continuous time elapsed from the last intervention, formally one can write it as follows.

$$e_p(k) = \frac{\lambda_p(k) - \overline{\lambda}_p(k)}{\lambda_p(k)} \tag{10}$$

$$IOI_p(k) = \sum_{i=1}^{k} [e_p(i-1)]^+ \cdot \Delta k \tag{11}$$

Notice that if the PM p is not fully loaded the positive part of e_p is equal to zero and the index does not increase. The presence of a threshold per each PM, $\overline{\lambda}_p(k)$ allows the data center owner to distinguish between machines that should be more loaded and others that should have more spare capacity, for example because they are hosting gold VMs and must provide a prescribed quality of service. If a migration occurs from source p at time k, the integrated overload index $IOI_p(k)$ is reset to zero. If the PM is still overloaded, its IOI will tend to grow, while if the migration solved the overload problem, the index will remain zero until a new overload condition occurs.

Apparently, the choice of $\overline{\lambda}_p(k)$ affects the number of overload detections. A wrong choice of $\overline{\lambda}_p(k)$ would lead to an unnecessary number of migrations, thus to bad performance. In the following we are describing a simple technique able to adapt its value to the actual load present on the PMs, aimed at spreading the load across the data center.

On the basis of the chosen overload index, one must set a threshold $\overline{OI}_p \in \mathbb{R}^+$. Once the threshold is hit by the chosen overload index, an overload situation is detected, and a migration is needed. Thus, the migration manager needs to decide which is the VM to migrate and where to migrate it.

The choice of the threshold $\overline{OI}_p$ decides for how much time the PM can stand an overload situation. The lower the $\overline{OI}_p$, the more reactive the system is to detect the need of a migration. The higher the $\overline{OI}_p$, the fewer migrations will occur.

C. Proposed policies

We use the described load monitor to understand when one should intervene, migrating a VM. The migration manager needs to select three different objects: the source PM, the VM to be migrated, and the destination PM. Ideally, the selection should be done so that the migration results in a reduction (and possibly in the complete recovery) of the overload. A migration is therefore identified by the tuple $\mathscr{M} = (k, s, w, d)$, where $k \in \mathbb{N}$ is the time index of the migration, $s \in \mathbb{P}$ is the source PM, $w \in \mathbb{V}$ is the VM to be migrated and $d \in \mathbb{P}$ is the destination PM.

We propose four different policies.

a) Random: For the first policy, we set the relocation threshold $\overline{OI}_p$ for $p \in \mathbb{P}$ as $\overline{OI}_p = 1.1 \cdot c_p$. If at time k at least one of the overload indexes $IOI_p(k)$ is greater than the corresponding threshold, we perform a migration. The migration manager selects a random PM s among the set of PMs where $IOI_p(k) > \overline{OI}_p$ as the migration source. It also randomly selects w among the VMs currently hosted on s to be migrated. The destination PM $d \in \mathbb{P} \setminus \{s\}$, is the PM that has less load.

$$d = \arg\min_{p \in \mathbb{P} \setminus \{s\}} \lambda_p(k) \tag{12}$$

Notice that this is not supposed to be fair, since less load does not mean more spare capacity. In the case where all the PMs have equal capacity, this policy is supposed to be non-optimized but fair, while in the heterogeneous case, not only the solution is non-optimized, but also fairness is not guaranteed.

b) Volume-aware (VA): As a second policy, we take into account the PMs' actual volume. As done for the random policy, we use the relocation threshold $\overline{OI}_p$ and randomly select a source PM s among those that satisfy the condition $IOI_p(k) > \overline{OI}_p$. For each PM p we select the couple (w,d) so that

$$(w,d) = \arg\max_{v \in \mathbb{V}, p \in \mathbb{P} \setminus \{s\}} (AAW_p(k) - L(k,v,s) \cdot V_v(k)) \tag{13}$$

This means that the policy selects the VM that is filling the biggest gap among the ones that could possibly be migrated. This policy is supposed to exploit the information about the heterogeneity of the PMs and be more fair with respect to the additional load they can stand.

c) Volume-aware with setpoint normalization (VA_x): As a third policy we discuss an extension of the second one. We compute the migration as in (13) but we try to equalize the load. For this, we consider a time varying setpoint $\overline{\lambda}_{p,\text{varying}}(k)$ and we update the setpoint every x time steps using a padding q and encouraging each PM to host the average load times a padding constant, which for example could be set as $q = 1.1$.

$$\overline{\lambda}_{p,\text{varying}}(k) = q \cdot \frac{1}{n_p} \cdot \sum_{i \in \mathbb{P}} \lambda_i(k). \tag{14}$$

This is supposed to work very well in homogeneous data center, where the load should be equalized among the different PMs. Alternatively, one can normalize the setpoints taking into account the heterogeneity of the data center, therefore substituting (14) with the following.

$$\overline{\lambda}_{p,\text{varying}}(k) = q \cdot \frac{c_p}{\sum_{j \in \mathbb{P}} c_j} \cdot \sum_{i \in \mathbb{P}} \lambda_i(k). \tag{15}$$

d) Migration likelihood with setpoint normalization (ML_x): In this fourth strategy, we select all the VMs hosted on PMs that are currently overloaded computing the set of VMs that could possibly be migrated $\mathbb{V}_m$. We denote with $\mathbb{P}_m$ the set of overloaded PMs, which is the set of PMs that satisfy $IOI_p(k) > \overline{OI}_p$. We select the VM to be migrated and the source PM by computing

$$(w,s) = \arg\min_{v \in \mathbb{V}_m, p \in \mathbb{P}_m} \frac{(L(k,v,p)V_v(k) + AAW_p(k))}{-\psi_{\pi_v}} + \psi_{\pi_v} \cdot \mu_v(k) \tag{16}$$

where ψ_{π_v} is a constant depending on the VM plan. We use 2 for a GOLD VM, 1.5 for a SILVER VM, 1.2 for a BRONZE and 1 for a BASIC one. The idea is that we select the machine trying to minimize the distance between the volume freed on the corresponding PM that would make PM non-overloaded. At the same time we penalize having too many migrations and the choice of machines that have more expensive plans. The destination d is selected as the one having higher spare volume, *i.e.*,

$$d = \arg\max_{p \in \mathbb{P} \setminus \{s\}} AAW_p(k). \tag{17}$$

D. Reference policy

In Section IV the policies presented above are compared with the migration policy proposed in [26], i.e., Sandpiper. In Sandpiper, an overload is flagged if at least κ out the n most recent observations and the next predicted value exceed a threshold. The prediction is performed using a one-step-ahead predictor of an $AR(n)$, i.e., autoregressive model of n-th order. In [26] it is suggested to select $\kappa = 3$ and $n = 5$, and a CPU utilization threshold for a PM of $\overline{\lambda}_p = 0.75 c_p$.

Then, in [26] the volume of a VM $v \in \mathbb{V}$ is defined as

$$V_v^{Sandpiper} := \frac{1}{1 - \%\ell_v^\circ} \cdot \frac{1}{1 - \%m_v^\circ} \tag{18}$$

where $\%\ell_v^\circ$ and $\%m_v^\circ$ are respectively the percentage of requested load and memory with respect to the PM on which v is running. Analogously, the volume of a PM $p \in \mathbb{P}$ is defined as

$$W_p^{Sandpiper} = \frac{c_p}{c_p - \sum_{v \in \mathbb{V}} L(k,v,p)\ell_v^\circ} \cdot \frac{M_p}{M_p - \sum_{v \in \mathbb{V}} L(k,v,p) m_v^\circ} \tag{19}$$

The authors also define the *volume-to-size ratio* (VSR) of a VM as the ratio between the volume of v and its memory requirement m_v°.

Whenever a migration is detected in a PM $p \in \mathbb{P}$ the following steps are performed.

1) The VMs in p are ordered by decreasing VSR.
2) The VM with the highest VSR is migrated to the least loaded PM, i.e., the PM with the least value of $W_p^{Sandpiper}$.
3) If the least loaded PM does not fit the selected VM, then the next least loaded PM is considered.
4) If no match is found for the selected VM, the next VM with the highest VSR is considered.

The described procedure is performed for all the PMs that are overloaded in the system.

IV. RESULTS

To produce comparable results, we implemented a simulator where we can execute the proposed policies on the same case study. The simulator is available as open source software[1] and can be reused by other researchers to compare their own policies on the same scenarios. All the sources of randomness in the simulator have been customized so as to produce the same sequence for the same experiment. Here we briefly describe our comparison indexes and then we discuss our experimental results.

[1] https://github.com/cloud-control/vm-migration-sim

A. Comparison indexes

In order to assess the quality of the proposed techniques, we here consider some indexes accounting for different aspects.

1) First of all we consider the Integral Square Error (ISE) of overload over a finite horizon T, which is defined as

$$ISE = \max_{p \in \mathbb{P}} \sum_{k=0}^{T} \left(\left(\lambda_p(k) - \overline{\lambda}_p(k) \right)^+ \right)^2 \Delta k. \tag{20}$$

2) Then, we consider the *number of migrations* per VM μ_v, $v \in \mathbb{V}$. In particular we are going to analyze the Empirical Cumulative Distribution Functionss (ECDFs) of μ_v for all $v \in \mathbb{V}$.
3) Another aspect that must be taken into account is the *inter-arrival migration time* for a single VM τ_v, $v \in \mathbb{V}$, i.e., the time elapsed between two subsequent migrations of a VM. Migrating a VM too often is very costly in terms of performance, and too frequent migrations leave few time to the VM to deliver its service. Thus, low values of τ_v should be avoided. For the sake of simplicity, we here show only the inter-arrival migration time of the VM that has the highest μ, i.e., that has been migrated the most.

B. Experimental results

In the following we present two case studies. The first one is a small size data center, while the second one is a larger one. The small data center models a private cloud, hosting only a few services. The larger data center models a large company facility.

1) Small data center: The small data center is composed of a set of three PMs, $\mathbb{P} = \{p_1, p_2, p_3\}$, each of them having 8 cores and 8GB of memory, and have a frequency of 2GHz. On top of these three PMs, eight VMs are deployed. The first two VMs, v_1 and v_2 have a GOLD plan, three VMs, v_3, v_4 and v_5 have a SILVER plan, while the remaining three v_6, v_7 and v_8 have a BASIC plan. The nominal loads are defined as follows: $\ell_1^\circ = 4$, $\ell_2^\circ = 2$, $\ell_3^\circ = 4$, $\ell_4^\circ = 3$, $\ell_5^\circ = 2$, $\ell_6^\circ = 6$, $\ell_7^\circ = 2$, $\ell_8^\circ = 2$. The VMs have similar memory demands: for each VM $v \in \mathbb{V}$, $m_v^\circ = 1$GB. At the beginning of the simulation, all the VMs are deployed on p_1. The data center is simulated for 10000 steps.

During the simulation, we assume that each VM v demands a varying amount of CPU and memory, that are normally distributed with average equal respectively to $0.75 \cdot \ell_v^\circ$ and $0.75 \cdot m^\circ$ and standard deviation equal 0.25.

Figure 2 shows the ISE obtained with the different migration strategies, as defined by (20). As can be seen, the introduced policies achieve a lower ISE value with respect to the reference algorithm (Sandpiper). The policy that achieves the lower ISE is the volume-aware with setpoint normalization, where the normalization is performed every five steps (VA_5). The migration likelihood with setpoint normalization every five steps (ML_5) strategy and the volume-aware policy

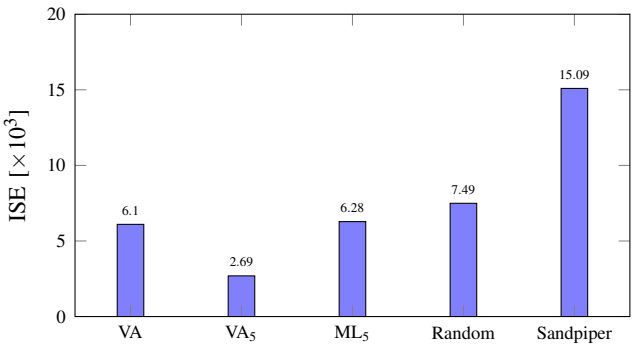

Fig. 2: ISE for each algorithm.

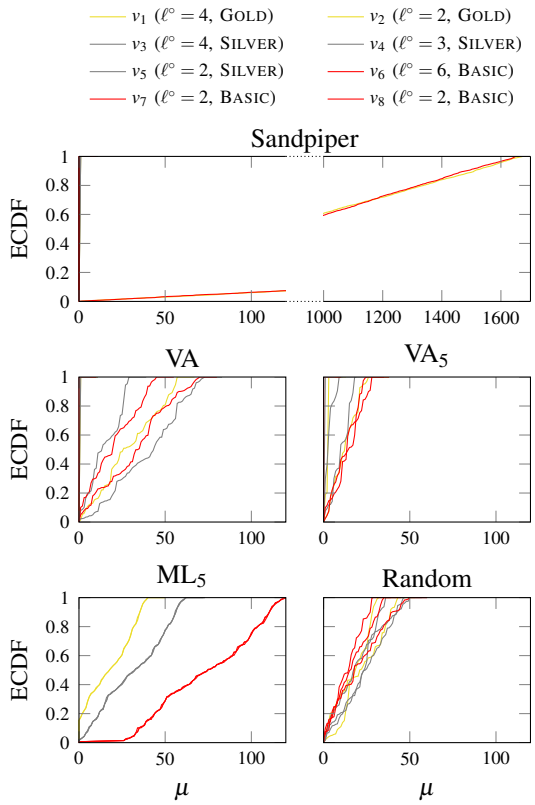

Fig. 3: ECDFs of the number of migrations for each VM.

without normalization (VA) achieve a low ISE compared to the Random policy and Sandpiper.

Figure 3 shows the ECDFs of the number of migrations μ_v for each $v \in \mathbb{V}$ — lines represent the probability that the corresponding VM was migrated less than or equal to a certain amount of times (on the x-axis). The VMs with a GOLD plan are represented with yellow lines, the SILVER with gray lines and the BASIC with red lines. While the Random and the VA policies treat all VMs equally, the ML policy distinguishes between different plans, migrating the GOLD and SILVER VMs less than the BASIC ones. While this is a desirable property, it should also be noted that the number of total migrations increases due to the additional requirement of plan differentiation. The plot of the ECDFs

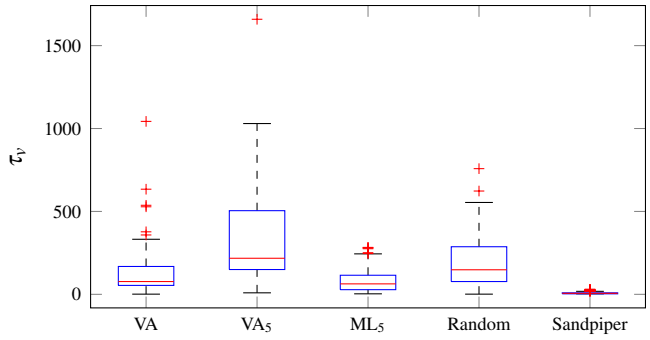

Fig. 4: Box plot of the distribution of inter-arrival migration times.

of Sandpiper shows that the number of migrations performed with this technique exceeds very much the numbers obtained with the proposed techniques. Moreover, two VMs are continuously migrated, a GOLD and a BASIC one, while the others experience a lower number of migrations.

Finally, Figure 4 shows the box plots of the inter-arrival migration times of the VM that was migrated the most during the simulation for each of the considered algorithms. Higher values of the inter-arrival time are preferable, while lower values indicate that the VM was migrated too frequently. Sandpiper is the algorithm that shows the worst performance, while VA_5 has the best distribution of inter-arrival times.

2) Large data center: For the second experiment we model a public cloud provider infrastructure. The data center is composed of 150 homogeneous PMs, each having 16 cores, 8GB of memory, and a frequency of 2GHz. We deploy 400 VMs, of which 30 have a GOLD plan with a CPU demand of $\ell^\circ_g = 2$, $g \in \{1,\ldots,30\}$, 70 have a SILVER plan with $\ell^\circ_s = 1$, $s \in \{31,\ldots,100\}$, 100 have a BRONZE plan with $\ell^\circ_b = 6$, $b \in \{101,\ldots,200\}$, and 200 have a BASIC plan with $\ell^\circ_r = 8$, $r \in \{201,\ldots,400\}$. The VMs have similar memory demands: for each VM $v \in \mathbb{V}$, $m^\circ_v = 1$GB. At the beginning of the simulation, the VMs are uniformly random distributed among the PMs. The data center is simulated for 10000 steps.

Figure 5 shows the ISE for the large data center experiment. The introduced policies achieve comparable values of ISE with respect to Sandpiper, with the exception of ML which performs worse. As for the number of migrations, however, VA and VA_5 show remarkable improvements (see Figure 6). On the other hand, none of the proposed policies distinguishes between the different kind of plans, except for ML_5.

Finally, Figure 7 shows the box plots of the inter-arrival migration times, confirming that VA and VA_5 behave better than the other algorithms, migrating VMs less frequently.

V. CONCLUSION AND FUTURE WORK

VM migration is an important problem in modern data centers and the migration strategy greatly affects the data center performance. Migrations can be performed to optimize the behavior of the data center in many different dimensions. A possibility is to compact the load as much as possible to

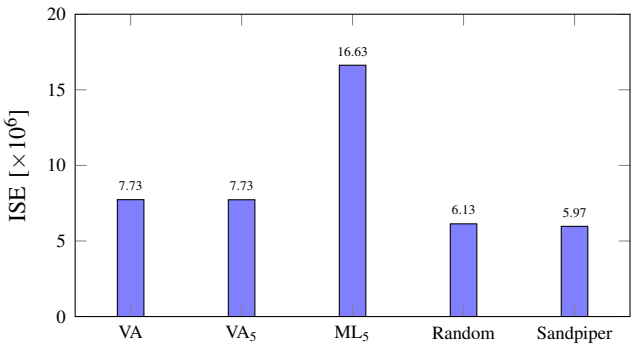

Fig. 5: ISE for each algorithm large data center.

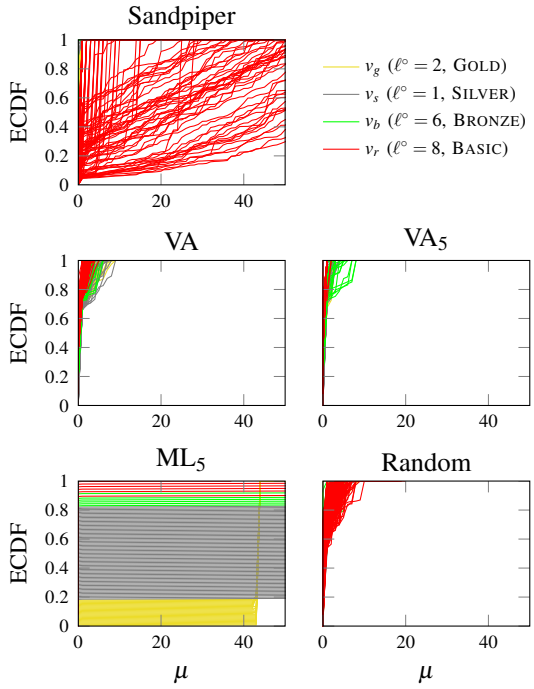

Fig. 6: ECDFs of the number of migrations for each VM for the large data center.

turn off the greatest amount of PMs possible. Another option is to equalize the load among the different PMs, to reject external disturbances in the best possible way. In this paper, we defined some policies that can be used for load balancing and compared them to one of the most known migration policies, Sandpiper, by means of a simulator. Our simulation results show that it is possible to define policies that behave better with respect to some optimization criteria (avoid the migration of VMs that were purchased with expensive plans, or follow a balancing setpoint).

In the future, we plan to extend the set of policies, introducing a policy specially designed for consolidation to enable the possibility of switching between different stages: Once the migration manager has figured out the number of PMs that are needed to serve a specific load, it is possible to switch to one of the load-balancing policies defined in this paper. We also plan to implement the proposed policies

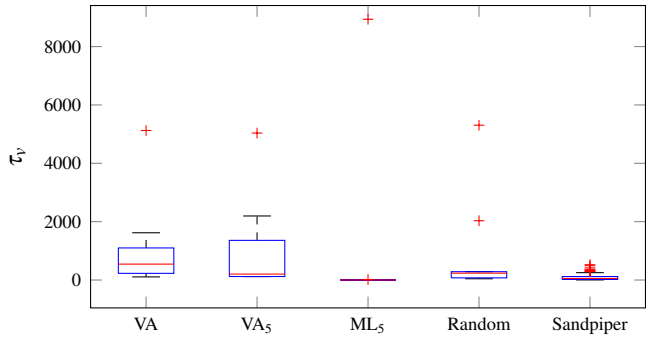

Fig. 7: Box plot of the distribution of inter-arrival migration times for the large data center.

in a cloud environment and test them in a private cloud infrastructure.

REFERENCES

[1] E. Arzuaga and D. R. Kaeli. Quantifying load imbalance on virtualized enterprise servers. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, WOSP/SIPEW '10, pages 235–242, New York, NY, USA, 2010. ACM.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[3] 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.

[4] N. M. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti. VM placement strategies for cloud scenarios. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, pages 852–859, Washington, DC, USA, 2012. IEEE Computer Society.

[5] L. Chen, H. Shen, and K. Sapra. Distributed autonomous virtual resource management in datacenters using finite-markov decision process. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 24:1–24:13, New York, NY, USA, 2014. ACM.

[6] L. Chen, H. Shen, and K. Sapra. RIAL: Resource intensity aware load balancing in clouds. In *INFOCOM, 2014 Proceedings IEEE*, pages 1294–1302, April 2014.

[7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[8] A. Corradi, M. Fanelli, and L. Foschini. VM consolidation: A real case based on openstack cloud. *Future Gener. Comput. Syst.*, 32:118–127, Mar. 2014.

[9] T. C. Ferreto, M. A. Netto, R. N. Calheiros, and C. A. De Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems*, 27(8):1027–1034, 2011.

[10] D. Gmach, J. Rolia, and L. Cherkasova. Selling T-shirts and time shares in the cloud. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 539–546, Washington, DC, USA, 2012. IEEE Computer Society.

[11] S. Hacking and B. Hudzia. Improving the live migration process of large enterprise applications. In *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '09, pages 51–58, New York, NY, USA, 2009. ACM.

[12] D. Haikney, S. Mullen, and J. Walker. Virtual machine migration, June 19 2014. US Patent App. 13/781,581.

[13] M. R. Hines, U. Deshpande, and K. Gopalan. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev.*, 43(3):14–26, July 2009.

[14] H. Hoffmann and M. Maggio. PCP: A generalized approach to optimizing performance under power constraints through resource management. In *11th International Conference on Autonomic Computing (ICAC 14)*, pages 241–247, Philadelphia, PA, June 2014. USENIX Association.

[15] J. Kelbley and M. Sterling. *Virtual Machine Migration*, pages 95–125. Wiley Publishing, Inc., 2010.

[16] L. Kou and G. Markowsky. Multidimensional bin packing algorithms. *IBM Journal of Research and Development*, 21(5):443–448, Sept 1977.

[17] Y. C. Lee and A. Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *J. Supercomput.*, 60(2):268–280, May 2012.

[18] H. Liu, H. Jin, C.-Z. Xu, and X. Liao. Performance and energy modeling for live migration of virtual machines. *Cluster Computing*, 16(2):249–264, 2013.

[19] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 248–259, New York, NY, USA, 2011. ACM.

[20] M. Nelson. Virtual machine migration, Jan. 27 2009. US Patent 7,484,208.

[21] M. Nelson, B.-H. Lim, G. Hutchins, et al. Fast transparent migration for virtual machines. In *USENIX Annual Technical Conference, General Track*, pages 391–394, 2005.

[22] F. L. Pires and B. Barán. Multi-objective virtual machine placement with service level agreement: A memetic algorithm approach. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, UCC '13, pages 203–210, Washington, DC, USA, 2013. IEEE Computer Society.

[23] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. *SIGPLAN Not.*, 46(7):111–120, Mar. 2011.

[24] L. Tomás and J. Tordsson. Improving cloud infrastructure utilization through overbooking. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, pages 5:1–5:10, New York, NY, USA, 2013. ACM.

[25] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe. CloudNet: Dynamic pooling of cloud resources by live WAN migration of virtual machines. *SIGPLAN Not.*, 46(7):121–132, Mar. 2011.

[26] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, volume 7 of *NSDI*, pages 229–242, 2007.

[27] Z. Xiao, W. Song, and Q. Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1107–1117, June 2013.