Exploiting VM Migration for the Automated Power and Performance Management of Green Cloud Computing Systems ⋆

Marco Guazzone, Cosimo Anglano, and Massimo Canonico

Department of Science and Innovation Technology
University of Piemonte Orientale
Alessandria, Italy
{marco.guazzone,cosimo.anglano,massimo.canonico}@mfn.unipmn.it

Abstract. Cloud computing is an emerging computing paradigm in which "Everything is as a Service", including the provision of virtualized computing infrastructures (known as *Infrastructure-as-a-Service* modality) hosted on the physical infrastructure, owned by an infrastructure provider. The goal of this infrastructure provider is to maximize its profit by minimizing the amount of violations of *Quality-of-Service* (QoS) levels agreed with its customers and, at the same time, by lowering infrastructure costs among which energy consumption plays a major role. In this paper, we propose a framework able to automatically manage resources of cloud infrastructures in order to simultaneously achieve suitable QoS levels and to reduce as much as possible the amount of energy used for providing services. We show, through simulation, that our approach is able to dynamically adapt to time-varying workloads (without any prior knowledge) and to significantly reduce QoS violations and energy consumption with respect to traditional static approaches.

Keywords: Cloud computing, Green computing, SLA

1 Introduction

The *Cloud Computing* [1] paradigm has enabled various novel modalities of service provisioning. Among them, particularly important is the *Infrastructure-as-a-Service* (IaaS) modality, whereby an *Infrastructure Customer* (IC) may provision his/her own virtual computing infrastructure, on which (s)he can deploy and run arbitrary software, on top on the physical infrastructure owned by an *Infrastructure Provider* (IP). Typically, ICs deploy on their virtual infrastructure multi-tier distributed applications consisting in an ensemble of *Virtual Machines* (VMs), each one hosting one or more application components together with the corresponding operating environment (i.e., OS and software libraries, just to name a few).

Usually, the IC and the IP agree on *Quality-of-Service* (QoS) levels that the IP commits to deliver to applications, which are defined in terms of suitable low-level temporal

⋆ This work was supported in part by the Italian Research Ministry under the PRIN 2008 *Energy eFFIcient teChnologIEs for the Networks of Tomorrow* (EFFICIENT) project.

and performance metrics commonly referred to as *Service Level Objectives* (SLO). This agreement, commonly referred to as *Service Level Agreement* (SLA), usually states that the IC accepts to pay a certain amount of money for using the infrastructure, and that the IP accepts to pay a money penalty for each violation of the SLA.

A critical issue that the IP must solve in order to maximize revenues is the amount of physical resources that must be allocated to each application. On the one hand, the allocation of an insufficient amount of resources may indeed yield a large number of SLA violations, with consequent losses due to corresponding money penalties. On the other hand, the allocation of an excessive amount of resources typically results in increases of energy costs, caused by a larger number of switched-on resources. These costs represent, at the moment, a large fraction of the *Total Cost of Ownership* (TCO) of the infrastructure [2], and are expected to further grow in the near future. Ideally, therefore, the IP should be able to allocate to each IC application no more than the minimum amount of resources resulting in no SLA violation, in order to keep the energy consumption to a bare minimum.

The standard approach generally adopted to allocate physical resources to the VMs of an application consists in statically assigning to each one of them enough capacity to fulfill the SLAs under the hypothesis that the workload will exhibit a specific intensity. This approach, however, fails to provide satisfactory results in realistic settings, because of the following factors:

- The characteristics of the workload are rarely known in advance, so it is very hard (if not impossible) to estimate its typical intensity. The resulting estimation errors lead to either under-provisioning (with consequent SLA violations) or over-provisioning of resources (with consequent energy waste).
- The intensity of the workload typically changes over time, so even if (unrealistically) the value of the intensity could be estimated without errors, the capacity allocated to the VMs should be dynamically adapted to these changes. This adaptation, however, may not be possible for a VM if the physical resource on which it is running has no spare capacity left, an event that is likely to occur in realistic settings where each physical resource is typically shared among many competing VMs, that usually exhibit different and conflicting resource utilization profiles.
- The VMs of the same application usually span different physical resources, that are all involved by the above adaptation, with the consequence that changes in the allocation of capacity to a given application create a domino effect involving a large set of other applications.

In order to tackle the problem of allocating the resources of a cloud physical infrastructure in face of the above issues, in [3] we proposed a distributed, time-hierarchical resource management framework that aims at simultaneously minimize SLA violations and energy consumption for multi-tier applications characterized by time-varying workloads whose characteristics are unknown in advance.

The framework encompasses three distinct levels, that operate on different system components at different time-scale, namely:

- an *Application Manager* for each application, that exploits control-theoretic techniques to monitor its SLOs, and dynamically determine the physical resource share that must be allocated to each of its VMs in order to meet the corresponding SLAs;

– a *Physical Machine Manager* for each physical resource, that multiplexes the total amount of physical capacity among all the VMs allocated on that resource;
– a *Migration Manager*, that monitors the SLOs of all applications and the overall energy consumption, and decides whether it is appropriate to migrate some VMs from the physical machines where they are allocated to other ones, in order to meet their SLAs and save energy.

In [3], we focused on the design and evaluation of the Application Manager and the Physical Machine Manager, and we showed that, when the workload is such that each physical machine has enough capacity to accommodate the simultaneous needs of all the VMs allocated on it, our framework results both in better performance and lower energy consumption than alternative static approaches.

This paper focuses instead on the Migration Manager. In particular, we present its architecture, the algorithms it uses, and a performance evaluation study carried out by means of *discrete-event simulation* [4], whereby we assess its ability to respect SLAs and save energy by comparing the performance attained by our framework when migration is enabled against those attained when migration is disabled, and against various static resource management approaches.

The rest of this paper is organized as follows. In Sect. 2, after a brief recap of our framework, we describe the Migration Manager. In Sect. 3, we present an experimental evaluation of our framework and show its effectiveness on minimizing SLO violations and energy consumption. In Sect. 4, we compare our approach with some recent related work. Finally, in Sect. 5, we conclude this paper and present future works.

2 The Resource Management Framework

The architecture of our framework includes a certain number of multi-tier applications which have to be deployed on a cloud infrastructure. Every application tier is deployed in a separate VM, which in turn is placed on one of the available Physical Machines (PMs). The core of our framework is the *Resource Manager*, which continuously monitors the performance of each deployed application and suitably acts on the system in order to maintain application performance goals and, at the same time, to minimize the energy consumption of computing resources.

We assume that the SLOs of each application are known, and are expressed in terms of a specific PM that we called *reference machine*. [1] It is responsibility of the Resource Manager to appropriately scale SLOs according to the capacity of physical resources belonging to the PMs where each application tier is actually run. To do so, we assume that the relative computing power (i.e., the measure of how much a physical resource is more powerful than another one) of two PMs of the same category can be expressed by a simple proportional relationship between their capacities. This means that if a resource has capacity of 10, it will be able to serve requests at a service rate double than a resource with capacity of 5.

[1] This choice appears to be natural, as (1) application performance generally vary according to the capacity of physical resources assigned to that application, and (2) physical resources inside cloud computing systems are usually heterogeneous.

In order to reduce energy consumption and achieve application performance targets, the Resource Manager exploits virtualization technologies and control-theoretic techniques. On the one hand, by deploying each application tier inside a separate VM, virtualization provides both a runtime isolated environment and a mean for dynamically provisioning physical resources to virtualized applications so that an effective use of physical resources can be achieved. On the other hand, control theory provides a way for enabling computing systems to automatically manage performance and power consumption, without human intervention. Thus, the Resource Manager accomplishes its goal by dynamically adjusting the fraction of the capacity of PMs assigned to each VM, and, if needed, by migrating one or more VMs into other and more appropriated PMs (possibly, by turning on or off some of them). Specifically, it acts on the cloud infrastructure via a set of independent components, namely *Application Manager*, *Physical Machine Manager*, and *Migration Manager*.

In the rest of this section we describe the Migration Manager, which is the focus of this paper. For the other two components, the interested reader may refer to [3] for a more thorough description. We just want to recall that the Application and the Physical Machine Managers act on a short-term time scale, and attempt to allocate, to the various tiers of each application, the share of physical resources they need to meet their SLOs.

It is worth noting that, although our framework is general enough to deal with any type of physical resource and performance metric, for the sake of simplicity, in this paper we restrict our focus to the CPU as the type of shared physical resource, and on the application-level response time, as the SLO performance metric.

2.1 The Migration Manager

The purpose of the Migration Manager is to find the placement of the currently running VMs on the PMs that results in the minimization of both SLO violations and energy consumption. This objective is achieved by monitoring application performance and by recomputing, if needed, a new VM placement that simultaneously results in the fulfillment of SLOs and in the lowest energy consumption among all the candidate ones. The new placement is computed at the end of regularly-spaced time intervals henceforth referred to as *control intervals*.

Once the optimal allocation is computed, the following (non-mutually exclusive) actions can be triggered: (1) one or more PMs are powered on in order to fulfill current CPU demand, (2) one or more VMs are migrated to more suitable PMs, and (3) one or more PMs are powered off due to an excess of available CPU capacity with respect to current demand. The optimal allocation is computed by solving, at each control interval k, an optimization problem, whose mathematical formulation (the *mathematical program*) is shown in Fig. 1, where M and V denote the set of all PMs of the cloud infrastructure (included the ones currently powered off), and the set of all virtual machines that are currently powered on, respectively, and $x_i(\cdot)$, $y_{ij}(\cdot)$, and $s_{ij}(\cdot)$ are the decision variables. More details about the mathematical program are provided in [5].

Solution Algorithm The above mathematical program is a *Mixed-Integer Nonlinear Program* (MINLP) [6] which is known to be NP-hard [7]. Thus, we resorted to an approximate solution computed by means of a *Best-Fit Decreasing* (BFD) strategy, which

$$
\begin{aligned}
\text{minimize} \qquad & J(k) = \Bigg[\frac{1}{g_e} \sum_{i \in M} W_i(k) x_i(k) \\
& \quad + \frac{1}{g_m} \sum_{i \in M} \sum_{h \in M} \sum_{j \in V} \pi_{jhi}(k) y_{hj}(k-1) y_{ij}(k) && \text{(1a)} \\
& \quad + \frac{1}{g_p} \sum_{i \in M} \sum_{j \in V} y_{ij}(k) \Big(s_{ij}(k) - \hat{s}_{ij}(k) \Big)^2 \Bigg] \\
\text{subject to} \qquad & x_i(k) \in \{0,1\}, && i \in M, && \text{(1b)} \\
& y_{ij}(k) \in \{0,1\}, && i \in M, j \in V, && \text{(1c)} \\
& s_{ij}(k) \in \big[0,1\big], && i \in M, j \in V, && \text{(1d)} \\
& \sum_{i \in M} y_{ij}(k) = 1, && j \in V, && \text{(1e)} \\
& y_{ij}(k) \leq x_i(k), && i \in M, j \in V, && \text{(1f)} \\
& x_i(k) \leq \sum_{j \in V} y_{ij}(k), && i \in M, && \text{(1g)} \\
& s_{ij}(k) \leq y_{ij}(k), && i \in M, j \in V, && \text{(1h)} \\
& s_{ij}(k) \frac{C_i}{\overline{C}} \geq y_{ij}(k) \bar{s}_{t_j}^{\min}, && i \in M, j \in V, && \text{(1i)} \\
& \sum_{j \in V} s_{ij}(k) \leq s_i^{\max}, && i \in M, && \text{(1j)} \\
& \hat{u}_i(k) \leq u_i^{\max}, && i \in M. && \text{(1k)}
\end{aligned}
$$

Fig. 1. Resource management framework – Optimization problem for the Migration Manager.

attempts to place the largest number of VMs on the fewest number of PMs, while still preserving SLO constraints. More details can be found in [5].

3 Experimental Evaluation

In order to assess the capability of our framework, we performed an extensive experimental evaluation using discrete-event simulation. To this end, we developed a C++ ad-hoc discrete-event system simulator, that is used to run our experiments.

We assess the performance of our framework by measuring, for each application, its response time and the number of SLO violations, as well as the amount of electrical energy (in Joule) consumed by the physical infrastructure. For each of these quantities, we compute the 95% confidence level by using the *independent replications* output analysis, where the length of each replication is fixed to $210,000$ ticks of simulated time, and the number of total replicas is fixed to 5. We use these fixed settings since the simulated system never reaches the steady-state due to the presence of applications that can start and stop in the middle of the simulation.

In our experiments, we use the 0.99^{th} quantile of application response time as SLO specification. This means that the infrastructure provider will pay penalties only when the percentage of SLO violations (i.e., the percentage of the number of times the observed response time is larger than the SLO value) is greater than 1% during a prescribed time interval.

To compute the SLO value for each application, we use a benchmark-like approach similar to the one described in [8,9] for application profiling: for each type of application and arrival process, we run a series of simulations (assigning to each tier exactly the amount of CPU capacity as defined by the reference machine specifications) and measure the 0.99^{th} quantile of the response time empirical distribution.

In the rest of this section, we first present the settings used to define the various simulation scenarios used for our study, and then we report and discuss the results obtained from the experiments.

3.1 Experimental Settings

The experimental scenarios are defined in terms of the configurations of the physical infrastructure and of the various applications, and of the specific settings for the parameters used by the various components of the framework.

Table 1. Experimental settings – Characteristics of the PMs and applications.

(a) Characteristcs of the PMs.

Group	No. of PMs	CPU Capacity	Power Model ω_0	ω_1	ω_2	ρ
M_1	3	1,000	86.7	119.1	69.06	0.400
M_2	2	2,000	143.0	258.2	117.2	0.355
M_3	2	3,000	178.0	310.6	160.4	0.311
M_4	2	4,000	284.0	490.1	343.7	0.462

(b) Characteristics of the applications.

Group	1^{st} Tier	2^{nd} Tier	3^{rd} Tier
A_1	0.060	0.06	0.06
A_2	0.030	0.06	0.03
A_3	0.015	0.03	0.06

Physical Infrastructure Configuration We consider a cloud infrastructure consisting of a set of 9 heterogeneous PMs, that, in turn, are divided into 4 groups, namely M_1, M_2, M_3, and M_4, in such a way that the PMs in the same group M_i have an identical CPU capacity that is i times larger than the one of the reference machine (that is assumed to be equal to 1,000).

The energy consumption of these machines is modeled as discussed in [10,11,12], whereby the power P absorbed by each machine is related to its CPU utilization u by means of the formula:

$$P = \omega_0 + \omega_1 u + \omega_2 u^{\rho} \tag{2}$$

where ω_0, ω_1 and ω_2 are model parameters. For each one of the machine classes, we use a different setting of the parameters in Equation (2) (that is used for all the machines in the same class), as reported in Table 1a (where we also report the capacity values and the number of PMs for each of the four machine groups). These values are estimated through a statistical regression analysis over data collected by the *SPECpower_ssj2008* benchmark [13].

Application Configuration We consider 6 three-tier applications divided into 3 groups, namely A_1, A_2, and A_3, such that applications in the same group have an identical behavior. For every application, each incoming service request arrives at the first tier that, after processing it, forwards it to the second tier where this process is repeated. Finally, after the processing in the third tier has taken place, the result of the service request is sent back to the respective client. These application classes differ from each other in the amount of processing time requested by each tier (expressed in terms of the computing power of the reference machine), as reported in Table 1b.

Each application is characterized by its workload, that consists in a stream of service requests, continuously generated by a population of users of unknown size, arriving according to a specific arrival process. In order to reproduce various operating conditions that may occur for real-world applications, we consider three different request arrival processes, namely:

- *Deterministic Modulated Poisson Process* (DMPP), to generate workloads exhibiting user behavioral patterns like daily-cycles of activity. In particular, we consider a three-state DMPP, henceforth denoted as DMPP$(\lambda_1, \lambda_2, \lambda_3, \tau)$, where λ_i, for $i = 1, \ldots, 3$, is the arrival rate of the Poisson process in state i, and τ is the deterministic state-residence time;
- *Pareto Modulated Poisson Process* (PMPP) [14] to generate self-similar workloads. In particular, we consider a two-states PMPP, from this time forth denoted as PMPP$(\lambda_1, \lambda_2, x_{\mathrm{m}}, \alpha)$, where λ_i , for $i = 1, 2$, is the arrival rate of the Poisson process in state i, and x_{m} and α are the minimum value and shape parameters of the Pareto distribution, respectively;
- *Markov Modulated Poisson Process* (MMPP) [15] to generate arrival processes exhibiting temporal burstiness [16]. In particular, we consider a two-states MMPP, henceforth denoted as MMPP$(\lambda_1, \lambda_2, \mu_1, \mu_2)$, where λ_i, for $i = 1, 2$, is the arrival rate of the Poisson process in state i, and μ_i, for $i = 1, 2$, is the state-transition rate when the process is in state i.

Starting from these arrival processes, we create four distinct application scenarios, each one including six instances of applications: three *persistent* instances (henceforth denoted as Per_1, Per_2, and Per_3) whose lifetime spans the entire simulation, and three *ephemeral* instances (henceforth denoted as Eph_1, Eph_2, and Eph_3) that arrive in the system at different instants (i.e., time $10,000$, $70,000$, and $130,000$, respectively), and leave $70,000$ time units after their arrival. The details of the four scenarios considered in our experiments, together with the SLOs of each application in each scenario, are reported in Table 2.

Table 2. Experimental settings – Arrival process parameters and SLO values of the applications. For each scenario, but the S-MIX one, the arrival process is the same as the one implied by the scenario name. In S-MIX, the arrival process is DMPP for A_1, MMPP for A_2 and PMPP for A_3.

Scenario	Application					
	A_1		A_2		A_3	
	Arrival	SLO	Arrival	SLO	Arrival	SLO
S-DMPP	$(1,5,10,3600)$	1.176	$(10,5,1,3600)$	0.612	$(5,10,1,3600)$	0.608
S-PMPP	$(5,10,1,1.5)$	1.245	$(5,10,1,1.5)$	0.655	$(5,10,1,1.5)$	0.624
S-MMPP	$(5,15,0.0002,0.002)$	4.001	$(5,15,0.0002,0.002)$	1.962	$(5,15,0.0002,0.002)$	1.935
S-MIX	$(5,10,1,3600)$	0.608	$(5,15,0.0002,0.002)$	1.935	$(5,10,1,1.5)$	0.624

Resource Manager Configuration The parameters characterizing the Resource Manager (see Sect. 2) include the ones associated to each Application Manager, Physical Machine Manager, and to the Migration Manager. Due to lack of space, we omit them from this paper. For more details, the interested reader can refer to [5].

3.2 Results and Discussion

In order to assess the effectiveness of the Migration Manager, we compare it with the following resource management static policies:

- STATIC-SLO: each VM (running a specific application tier) is assigned the amount of CPU capacity $Cap(SLO)$ needed to meet application SLOs, so that SLOs satisfaction is favoured over energy consumption reduction;
- STATIC-ENERGY: each VM is statically assigned a fixed amount of capacity that is 25% lower than $Cap(SLO)$. This is an energy-conserving approach since, by assigning less capacity than required, energy consumption reduction is given priority over SLOs satisfaction;
- STATIC-TRADEOFF: each VM is assigned a fixed amount of capacity that is 10% lower than $Cap(SLO)$, in order to seek a reasonable trade-off between energy consumption reduction and SLOs preservation.

It should be noted that all these policies are based on the rather unrealistic assumption that $Cap(SLO)$ is exactly known, whose knowledge requires the ability to perform measurement experiments during which each application is exposed to its workload for a suitable amount of time.

Moreover, in order to assess the benefits of VM migration with respect to situations where only the Application Manager and the Physical Machine Manager are present (as in our previous work [3]), we consider two different variants of our approach, denoted respectively as OUR-APPROACH-NM and OUR-APPROACH-M, which differ in that the former does not use the Migration Manager component (and hence does not employ VM migration).

Our comparison is performed by considering the following performance indices (for each one of them, we compute the 95% confidence intervals):

– *SLO violations*: the mean percentage of SLO violations for each application instance;
– *Total Energy* (TotEn): total amount of electrical energy (measured in kilowatt-hours, kWh) spent to serve all the requests received by applications.
– *Wasted Energy* (WastEn): total amount of electrical energy (measured in kilowatt-hours, kWh) spent to serve out-of-SLO requests (this amount of energy can be, therefore, considered as wasted to serve requests that will imply a money penalty for the provider).

Due to lack of space, we only report part of performed experiments. The interested reader can refer to [5] for a complete discussion of experimental evaluation. The results of the various scenarios are presented in two separate tables, namely Table 3 (for S-PMPP) and Table 4 (for S-MIX). In each table, every column reports the results obtained, by the various applications, under a specific resource management approach (an "n/a" value means that the use of the corresponding resource management approach made the simulation unable to converge).

Table 3. Experimental results – S-PMPP scenario. Energy consumption refers to a period of 210,000 simulated seconds (i.e., approximately 2.5 simulated days).

			Approach				
			STATIC-∗			OUR-APPROACH-∗	
			SLO	ENERGY	TRADEOFF	NM	M
SLO Violations	(%)	Per_1	0.58	19.60	2.57	0.58	0.59
		Eph_1	0.89	33.89	4.17	0.90	0.90
		Per_2	0.68	16.04	2.68	0.68	0.68
		Eph_2	0.82	9.33	3.24	0.83	0.82
		Per_3	0.53	15.62	2.37	0.53	0.53
		Eph_3	0.47	11.57	1.84	0.47	0.47
Energy Consumption	(kWh)	TotEn	109.28	116.46	118.72	107.31	98.27
		WastEn	0.70	20.65	3.23	0.69	0.63

By looking at the results reported in these tables, we can first observe that while both OUR-APPROACH-M and OUR-APPROACH-NM practically result in identical values of SLO violations for all the scenarios, the former always results in a more efficient usage of physical resources. As a matter of fact, OUR-APPROACH-M always results in lower values of TotEn metric: this means that the exploitation of VM migration allows to both save energy (lower TotEn values) and to better use the energy that is consumed (lower WastEn values).

Let us now compare the performance of OUR-APPROACH-M against those attained by the static policies.

As indicated by our results, STATIC-ENERGY and STATIC-TRADEOFF always result in higher values of SLO violations and energy consumption than OUR-APPROACH-M. Furthermore, STATIC-ENERGY can be considered worse than STATIC-TRADEOFF

Table 4. Experimental results – S-MIX scenario. Energy consumption refers to a period of 210,000 simulated seconds (i.e., approximately 2.5 simulated days).

			Approach				
			STATIC-∗			OUR-APPROACH-∗	
			SLO	ENERGY	TRADEOFF	NM	M
SLO Violations	(%)	Per_1	0.81	n/a	3.23	0.81	0.81
		Eph_1	0.79	n/a	3.14	0.79	0.79
		Per_2	0.87	n/a	20.86	0.84	0.82
		Eph_2	0.44	n/a	15.09	0.51	0.52
		Per_3	0.55	n/a	2.36	0.55	0.56
		Eph_3	0.65	n/a	2.75	0.65	0.66
Energy Consumption	(kWh)	TotEn	87.99	n/a	91.41	87.58	82.65
		WastEn	0.63	n/a	7.12	0.65	0.58

since it results in a moderate improvement of (unit) energy consumption at the price of much higher values of SLO violations. Therefore, we can conclude that our approach is able to satisfy SLOs for a greater number of requests with a lower energy consumption and, more importantly, without resulting in any penalty to be paid by the provider, than these two static approaches.

The comparison with the STATIC-SLO policy needs more attention. First of all, we can observe that OUR-APPROACH-M practically exhibits the same values of SLO violations than STATIC-SLO for all the scenarios. For this latter scenario, OUR-APPROACH-M results in a higher number of SLO violations only for the Per_1 application, while, for the remaining applications, this metric practically exhibits the same values.

If, however, we look at the efficiency-related metrics, we can observe that OUR-APPROACH-M always results in lower values of TotEn and WastEn than STATIC-SLO, indicating that the former is able to consume less energy and to use physical resources more effectively. Moreover, it is important to observe that STATIC-SLO, in addition to be unrealistic, requires an overcommitment of resources, whereby a larger fraction of CPU capacity is assigned to each VM regardless of the fact that this fraction will be actually used by the VM. As a result, this implies that the number of VMs that can be consolidated on the same PM is lower than those attained by our approach (that, instead, allocates to each VM just the fraction of CPU capacity it needs). Therefore, STATIC-SLO potentially requires, for a given number of VMs, a larger number of physical resources than the our approach one, thus yielding a larger energy consumption.

4 Related Works

The problem of dynamically managing physical resources of a cloud infrastructure in such a way to take into consideration both performance targets of hosted applications and power consumption of the infrastructure has been studied in the literature only recently.

In [17], a series of adaptive algorithms for dynamic VM consolidation are proposed and evaluated. Unlike this work, our solution, has a decentralized architecture

and, through the combination of online system identification and VM migration, is able to work with any type of workload.

In [18], a combined predictive and reactive provisioning technique is proposed, where the predictive component estimates the needed fraction of capacity according to historical workload trace analysis, while the reactive component is used to handle workload excess with respect to the analyzed one. Unlike this work, our solution, by means of online system identification, is potentially able to work with any type of workload without any prior knowledge.

In [19], an online resource provisioning framework is proposed for combined power and performance management as a sequential multi-objective optimization problem under uncertainty and is solved using limited lookahead control, which is a form of model predictive control [20]. The key difference between this work and our approach is in the type of the architectural design adopted: centralized for the former, and decentralized for the latter. It is important to note that authors also show, via trace-driven simulation, that the use of a feed-forward neural network can make their work potentially able to scale to large systems. Unfortunately, the lack of details makes us unable to perform any sort of comparison.

In [21], a two-layers hierarchical control approach is proposed. Even though this work shares some architectural similarity with our framework, there are two important differences; the first difference is in the way parameters of the black-box application model (i.e., ARX) are computed (i.e., through offline identification), while the second one is in the way CPU shares are assigned to application tiers (which is not well suited for situations where tiers of different applications are hosted on the same PM).

Finally, in [22], a two-levels control architecture is proposed, where, unlike our approach, the achievement of performance targets is always subjected to the reduction of power consumption; conversely, in our approach, the reduction of power consumption is constrained to the achievement of performance targets.

5 Conclusions

In this paper, we presented a framework for automatically managing computing resources of cloud computing infrastructures, in order to simultaneously satisfy SLO constraints and reduce system-level energy consumption, that exploits VM migration to obtain its goals. By means of simulation, we showed that, compared to traditional static approaches, our framework is able to dynamically adapt to time-varying workloads (with no prior knowledge) and, at the same time, to significantly reduce both SLO violations and energy consumption. Furthermore, we showed that VM migration actually results in significant performance and energy consumption improvements over situations where the same framework does not use VM migration.

There are several avenues of research that we plan to explore in the near future. First of all, we would like to extend the framework to also take into consideration other resource types (e.g., memory and disks) and system components (e.g. network routers and switches). Furthermore, we would like to extend the framework in such a way to enable it to manage federations of cloud infrastructures. Finally, we plan to integrate it into a cloud management middleware in order to test it under real operating conditions.

References

1. Weiss, A.: Computing in the clouds. netWorker **11**(4) (2007) 16–25
2. ENERGY STAR Program: Report to congress on server and data center energy efficiency. Technical report, U.S. EPA (Aug 2007)
3. Guazzone et al., M.: Energy-efficient resource management for cloud computing infrastructures. In: Proc. of the 3rd IEEE Int. Conf. on Cloud Computing Technology and Science (CloudCom'11). (2011)
4. Banks et al., J.: Discrete-Event System Simulation. 5th edn. Prentice Hall (2010)
5. Guazzone et al., M.: Exploiting VM migration for the automated power and performance management of green cloud computing systems. Technical Report TR-INF-2012-04-02-UNIPMN, University of Piemonte Orientale (April 2012)
6. Lee et al., J., ed.: Mixed Integer Nonlinear Programming. Volume 154 of The IMA Volumes in Mathematics and its Applications. Springer Science+Business Media, LLC (2012)
7. Jeroslow, R.: There cannot be any algorithm for integer programming with quadratic constraints. Oper. Res. **21**(1) (1973) 221–224
8. Yang et al., L.T.: Cross-platform performance prediction of parallel applications using partial execution. In: Proc. of the 2005 ACM/IEEE Conference on Supercomputing (SC'05). (2005)
9. Wood et al., T.: Profiling and modeling resource usage of virtualized applications. In: Proc. of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware (Middleware'08). (2008) 366–387
10. Beloglazov et al., A.: A taxonomy and survey of energy-efficient data centers and cloud computing systems. In Zelkowitz, M.V., ed.: Advances in Computers. Volume 82. Elsevier (2011) 47–111
11. Fan et al., X.: Power provisioning for a warehouse-sized computer. In: Proc. of the 34th Int. Symp. on Computer Architecture (ISCA'07). (2007) 13–23
12. Rivoire et al., S.: A comparison of high-level full-system power models. In: Proc. of the 2008 USENIX Conf. on Power Aware Computing and Systems (HotPower'08). (2008) 1–5
13. Standard Performance Evaluation Corporation: SPECpower_ssj2008 benchmark. Available: `http://www.spec.org/power_ssj2008`
14. Le-Ngoc et al., T.: A Pareto-modulated Poisson process (PMPP) model for long-range dependent traffic. Comput. Comm. **23**(2) (2000) 123–132
15. Fischer et al., W.: The Markov-modulated Poisson Process (MMPP) cookbook. Perform. Eval. **18**(2) (1993) 149–171
16. Mi et al., N.: Injecting realistic burstiness to a traditional client-server benchmark. In: Proc. of the 6th IEEE Int. Conf. on Autonomic Computing (ICAC'09). (2009) 149–158
17. Beloglazov et al., A.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. Concurrency Comput. Pract. Ex. Accepted for publication.
18. Gandhi et al., A.: Minimizing data center SLA violations and power consumption via hybrid resource provisioning. In: Proc. of the 2nd Int. Green Computing Conf. (IGCC'10). (2011)
19. Kusic et al., D.: Combined power and performance management of virtualized computing environments serving session-based workloads. IEEE Trans. on Netw. and Serv. Manag. **8**(3) (2011) 245–258
20. Camacho et al., E.F.: Model Predictive Control. 2nd edn. Springer (2004)
21. Xiong et al., P.: Economical and robust provisioning of *n*-tier cloud workloads: A multi-level control approach. In: Proc. of the 31st Int. Conf. on Distributed Computing Systems (ICDCS'11). (2011) 571–580
22. Wang et al., X.: Coordinating power control and performance management for virtualized server clusters. IEEE Trans. Parallel Distrib. Syst. **22**(2) (2011) 245–259