Energy-Efficient Resource Management for Cloud Computing Infrastructures

Marco Guazzone, Cosimo Anglano and Massimo Canonico
Department of Computer Science
University of Piemonte Orientale
Alessandria, Italy
Email: {marco.guazzone,cosimo.anglano,massimo.canonico}@mfn.unipmn.it

***Abstract*—Cloud computing is growing in popularity among computing paradigms for its appealing property of considering "Everything as a Service". The goal of a Cloud infrastructure provider is to maximize its profit by minimizing the amount of violations of Quality-of-Service (QoS) levels agreed with service providers, and, at the same time, by lowering infrastructure costs. Among these costs, the energy consumption induced by the Cloud infrastructure, for running Cloud services, plays a primary role. Unfortunately, the minimization of QoS violations and, at the same time, the reduction of energy consumption is a conflicting and challenging problem. In this paper, we propose a framework to automatically manage computing 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.**

I. Introduction

Cloud computing, a "*model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" [1], has recently rapidly grown in popularity [2]. A number of organizations are already benefiting of it by hosting and/or offering Cloud computing services. Among the various reasons behind this success, the most prominent ones are probably:

- *on-demand self-service*: a consumer can autonomously provision computing capabilities (e.g., computing power, storage space, network bandwidth), that is without requiring human interaction with the respective provider(s);
- *rapid elasticity*: the above capabilities may be dynamically resized in order to quickly scale up (to potentially unlimited size) or down in according to the specific needs of the consumer.

Together, these two properties are transforming the traditional way in which computing resources are used, shifting the focus from the classical vision under which (software and hardware) resources must be purchased before they can be used to run the applications of interest, to a novel one under which "Everything is a Service", and can be used in a *pay-as-you-go* modality. This service-centric vision permeates all the various components of the application stack, ranging from software (*Software as a Service (SaaS)*) to computing platforms (*Platform as a Service (PaaS)*), and – finally – computing infrastructure (*Infrastructure as a Service (IaaS)*).

The basic ingredient of Cloud Computing is the IaaS substrate, that enables service providers to provision the infrastructure they need for the delivery of their services without having to buy the resources composing them. Generally speaking, infrastructure providers rely on one or more data centers, comprising a suitable number of physical resources, and on the use of various *resource virtualization* technologies, whereby the same physical resource may be shared among multiple applications by deploying each of them into one or more *Virtual Machines* (VMs), each of which representing an isolated runtime environment.

It is evident that, in order for this paradigm to be successful, it must enable both the service and infrastructure provider to make profit out of the respective activities. Therefore, typically they agree on a prescribed set of service levels, commonly referred to as *Service Level Agreement* (SLA), that is a formal description of temporal, performance and economical constraints under which hosted services (e.g., enterprise applications) need to operate. It is commonly described in terms of *Service Level Objectives* (SLOs), which in turn define temporal and performance metrics for measuring the quality of service (e.g., the 99% of served user requests must have a response time no greater than a particular amount of seconds). [1] In other words, the service provider agrees on paying a certain amount of money for using the infrastructure, and the infrastructure provider agrees on providing enough resources to meet the service levels defined by the SLOs.

The decision of the amount of computing resources to allocate to a specific customer may have a critical impact on the revenues of the infrastructure provider. As a matter of fact, if insufficient resources are provided, the SLOs may be missed, and typically any SLO miss involves a money penalty for the IaaS provider, whose profit is consequently reduced. On the other hand, if the infrastructure provider opts for an over-provisioning of resources, a larger number of physical resources are used to host the same service, thus increasing the

[1] Various techniques to determine the SLOs corresponding to a given SLA have been published in the literature (e.g., [3], [4]).

Total Cost of Ownership (TCO), which comprises capital and administrative costs [5]. Thus, the ultimate goal for an IaaS provider is to maximize its profit by minimizing the number of such violations and, at the same time, by reducing the TCO.

This is a challenging problem because of the conflicting nature of the two aspects. Indeed, on one hand the achievement of SLOs would lead the provider to over-provision hosted services, while, on the other hand such over-provisioning would impact on TCO by investment, operating and energy consumption costs.

It has been argued [6] that energy costs are among the most important factors impacting on TCO, and that this influence will grow in the near future due to the increase of electricity costs. Therefore, the reduction of operational costs is usually pursued through the reduction of the amount of energy absorbed by the physical resources of the data center.

Various techniques already exist that aim at reducing the amount of electrical energy consumed by the physical infrastructure underlying the IaaS substrate, ranging from energy-efficient hardware and energy-aware design strategies, to *server consolidation*, whereby multiple virtual machines run on the same physical resource [7]. Unfortunately, these techniques alone are not enough to guarantee application performance requirements because of the complexity of Cloud computing systems, where (1) system resources have to be dynamically and unpredictably shared among several independent applications, (2) the requirements of each application must be met in order to avoid economical penalties, (3) the workload of each application generally changes over time, (4) applications may span multiple computing nodes (e.g., multi-tier applications), and (5) system resources may be possibly distributed world-wide. Moreover, the inherently conflicting nature of energy and performance management, along with the complexity of Cloud computing systems, makes a manual or semi-automatic approach unsuitable, so that much of current research work is looking for coordinated and fully automated solutions.

In this paper, we tackle the problem of providing a fully automated solution to the problem of dynamically managing physical resources of a (possibly distributed) data center whose resources are used as substrate for IaaS platforms. We present a framework able to automatically manage physical and virtual resources of a Cloud infrastructure in such a way to maximize the profit of the IaaS provider by minimizing SLO violations while, at the same time, reducing the energy consumed by the physical infrastructure. Basically, we accomplish this goal by providing each application with the minimum amount of physical resource capacity needed to meet its SLOs, and by dynamically adjusting it according to various parameters, that include the intensity of its workload, the number of competing VMs allocated on the same physical resource, and their time-varying behavior induced by variations in the respective workloads. The rationale underlying this approach is that, in order to balance energy consumption and SLOs satisfaction, each application needs exactly the fraction of physical resource capacity as the one dictated by current operating conditions of the Cloud infrastructure (e.g., workload characteristics and physical resource utilization, just to name a few). As a matter of fact, on one hand, a greater amount of physical resource capacity would imply an increase of energy consumption without any benefit to the profit (i.e., to stay away from the performance target is essentially identical – in terms of positive income – to stay very close to such objective). On the other hand, a smaller fraction of physical resource capacity would increase the probability of incurring in a SLO violation.

The contributions of this paper may be summarized as follows:

1) we define a framework that combines short-to-medium term allocation decisions (by which the capacity of a given physical resource is assigned to the various VMs running on it) with long-term ones (by which a given VM may be migrated from one physical resource to another), and that is able to deal with multi-tier applications under very variable and unpredictable workloads;
2) our approach, unlike previous ones, does not require neither any prior knowledge nor the construction of off-line application models, but is able to dynamically adapt at run time;
3) we show, via *discrete-event system* (DES) simulation [8], that our solution is able to manage physical resources of a data center in such a way to significantly reduce SLO violations (with respect to a traditional approach in which resource capacity is statically allocated to individual VMs), and at the same time to strongly improve the energy-efficiency of the infrastructure (defined as the amount of energy used to serve a single application request).

Although the resulting 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 physical resource, and on the application-level response time, as SLO performance metric.

The rest of this paper is organized as follows. In Section II, we describe the architecture and design of our framework. In Section III, we provide an experimental evaluation of our framework and show its effectiveness on minimizing SLO violations and energy consumption. In Section IV, we compare our solution with some recent related work. Finally, in Section V, we conclude this paper and present future works.

II. The Resource Management Framework

In this section, we describe our resource provisioning framework that jointly manages both the power consumption of physical resources and the performance attained by the services they host. Firstly, we provide a high-level description of the architecture of the whole framework. Then, we focus on the design of the *Resource Manager*, which is the main component of our framework.

A. System Architecture

The goal of our framework is three-fold: (1) to provide automated resource management mechanisms and policies,

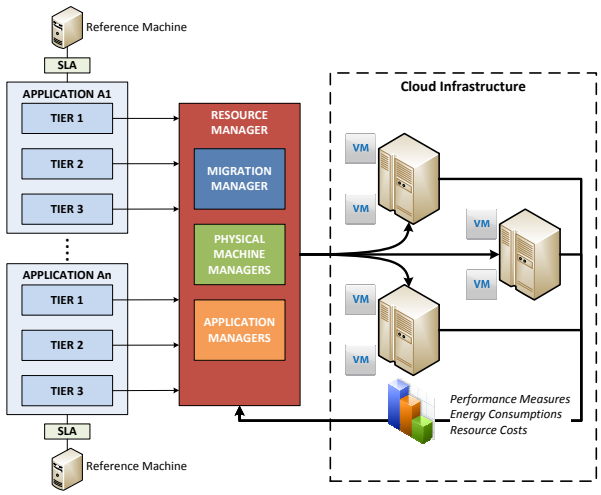

Fig. 1. Architecture of the proposed framework.

(2) to monitor and maintain application performance targets and (3) to reduce energy consumption in Cloud computing systems.

A high-level architecture of our framework is depicted in Fig. 1, where a certain number of user applications (on the left side) have to be deployed on a Cloud infrastructure (on the right side). The applications we considered are multi-tier; this choice does not limit the applicability of our framework since other type of applications (like high-performance computing applications) could always be modeled as single-tier applications. As shown in the figure, every application tier is deployed in a separate VM, which in turn is placed on one of the available physical machines. At the center of the figure, the *Resource Manager* 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 assumed that the SLO constraints of each application are known, and are expressed in terms of a specific physical machine that we called *reference machine* (e.g., in the Amazon EC2 terminology, this could be the equivalent of the "standard instance"). 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. It is responsibility of the Resource Manager to appropriately scale SLO constraints according to the capacity of physical resources belonging to the physical machines where each application tier is actually run. To do so, we assumed that the relative computing power of two physical resources of the same category (i.e., the measure of how much a physical resource is more powerful than another one) can be expressed by a simple proportional relationship between the capacity of the two resources. This means that if a resource has capacity equals to 10, it will be able to serve requests at a double service rate than a resource with capacity equals to 5.

In order to reduce energy consumption and achieve application performance targets, the Resource Manager combines virtualization technologies and control-theoretic techniques. On 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 physical resources assigned to each VM (hosting a particular application tier), and, if needed, by *migrating* one or more VMs into other and more appropriated physical machines (possibly, by turning on or off some of them). As shown in Fig. 1, the Resource Manager consists in a set of independent components that we called *Application Manager*, *Physical Machine Manager*, and *Migration Manager*.

Currently, only the Application and Physical Machine Managers have been implemented. For this reason, in the rest of this section we focus on the design of these two components. We just want to remark that the purpose of the Migration Manager (whose implementation is ongoing) is to monitor, at long-time scale, performance targets and energy consumption and to decide which VMs need to be migrated to suitable physical machines (possibly by turning on additional physical machines, for achieving better performance) and which physical machines can be turned off (to save energy).

B. Application Manager

The purpose of the Application Manager is to provide the controlled application with the needed amount of resource capacity in order to satisfy its SLO constraints. The Application Manager accomplishes its task by periodically performing the following actions: (1) it monitors the interested performance metrics, (2) it compares them with the related counterparts defined by the SLOs (associated to the controlled application), (3) it computes the amount of resource capacity each tier should obtain to meet its SLO constraints, and (4) it forwards these resource capacity demands to Physical Machine Managers associated to physical machines where each tier of the controlled application is running. Moreover, in order to cope with physical machine heterogeneity, at the beginning and the end of its periodic activity it converts actual resource demands (related to physical machines where tiers are currently running) to/from "reference" ones (stated in terms of the reference machine), respectively. As shown in Fig. 2, we designed the Application Manager as an *adaptive feedback controller* by using a *Self-Tuning Regulation* (STR) adaptation scheme [9]; in what follows, we provide a brief description of each component (mathematical details are out of scope of this paper). In the figure, the "target system" box (i.e., the controlled application) is modeled as an *AutoRegressive with eXogenous terms* (ARX) model [10], where system inputs (i.e., CPU shares) and outputs (i.e., tier residence times) are

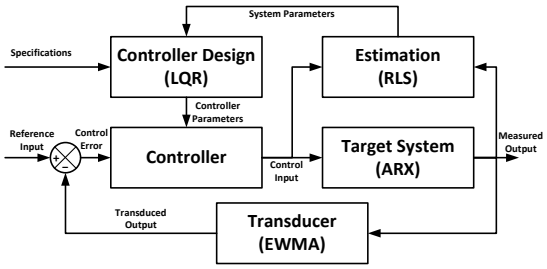

Fig. 2. Internal structure of the Application Manager.

restated as normalized deviations from an operating point [11]. The "estimation" box estimates on-line (i.e., at each control interval) ARX parameters, to adapt them to dynamic workload changes, by means of the *Recursive Least-Square* (RLS) algorithm with *exponential-forgetting*, which uses a *forgetting factor* for regulating the exponential decay of the influence of past observations over new ones. The "transducer" box represents an *Exponentially Weighted Moving Average* (EWMA) filter [12] used to smooth the observed system output (to prevent the controller to be too reactive in case of short peaks in the output), and for coping with low-intensity control periods (i.e., when no request leaving the application is observed). EWMA uses a *smoothing factor* for regulating the exponential decay of the influence of past observations over new ones. The "controller design" and "controller" boxes converts the ARX system model to an equivalent *state-space* representation and uses the infinite-horizon discrete-time *Linear Quadratic Regulator* (LQR) with output weighting control design [13] in order to compute (by minimizing a quadratic cost function) optimal system inputs given current system outputs; it is characterized by two weighting matrices representing the output weight and the control effort to apply during the optimization.

Once optimal CPU shares are computed, the Application Manager rescales them with respect to the CPU capacity of the physical machines where the associated tiers are currently running, and then forwards the computed shares to the Physical Machine Managers (described in the next section) controlling those physical machines.

C. Physical Machine Manager

The purpose of the Physical Machine Manager is to satisfy CPU share demands coming from those Application Managers which have tiers running on the controlled physical machine. There is one Physical Machine Manager for each physical machine of the Cloud infrastructure. It is important to note that since (1) the same physical machine may hosts VMs running tiers belonging to different applications, and (2) Application Managers work independently from each others, CPU share demands arriving at the Physical Machine Manager are generally uncorrelated, so that the aggregated CPU share demand may exceed the maximum CPU capacity of the controlled physical machine. Thus, the Physical Machine Manager has to arbiter among all incoming CPU share demands by adjusting them according to a given policy. In our current implementation, we designed the Physical Machine Manager according to a *proportional* policy, whereby adjusted CPU share demands are computed proportionally to the original demands. Specifically, assuming that a particular physical machine hosts n VMs, CPU shares are bounded in the $(0, D]$ real interval (with $0 < D \leq 1$), and denoting with $d_1, \ldots, d_n$ the incoming CPU share demands for the n VMs, the adjusted CPU share demands $\hat{d}_1, \ldots, \hat{d}_n$ will be computed according to the following formula:

$$\hat{d}_i = \frac{d_i}{\sum_{j=1}^{n} d_j} D \tag{1}$$

In the following section we show and discuss the results obtained by running various simulations for evaluating the efficacy of the just described framework.

III. Experimental Evaluation

In this section, we present the results of the experiments we conducted for evaluating the effectiveness of our proposed solution. Firstly, in Section III-A, we describe the experiments setup, then in Section III-B, we present and discuss the related results.

A. Experimental Setup

In order to assess the capability of the resource management techniques described in this paper, we performed an extensive experimental evaluation using discrete-event simulation. To this end, we developed a C++ ad-hoc discrete-event simulator, that was used to run our experiments.

1) Physical Infrastructure Configuration: We considered a set of homogeneous physical machines whose computing power is twice that of the reference machine. For all machines, the consumption of electrical energy was modeled by means of the power model described in [14], whereby the power P is related to the CPU utilization u by means of the formula $P = C_0 + C_1 u + C_2 u^r$. We estimated the values of the parameters C_0, C_1, C_2 and r through a statistical regression analysis over data collected by the *SPECpower_ssj2008* benchmark [15], and we set them to $C_0 = 143$, $C_1 = 258.2$, $C_2 = 117.2$, and $r = 0.355$.

The placement of each VM is performed at the beginning of the simulation with a static approach; each VM is randomly assigned to 1 of 5 physical machines, thus resulting in at most 2 VMs per physical machine.

2) Application Configuration: We considered a set of 3 distinct multi-tier applications, that we named A_1, A_2, and A_3, respectively. All these applications have 3 tiers: 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. The applications considered in our experiments differ from each other in the amount of processing time requested by each tier. In particular:

- A_1 is an application where all tiers have the same processing capacity, that has been set to 0.06 req/sec;

- A_2 is an application with a bottleneck in the second tier; more specifically, the first and third tiers have the same service rate set to 0.03 req/sec, while the second tier has a service rate set to 0.06 req/sec;
- A_3 is an application where tiers have decreasing processing capacity, and has been obtained by setting the service rates of the first, second, and third tiers to 0.015, 0.03, and 0.06 req/sec, respectively.

All the above service rates are expressed in terms of the computing power of the reference machine.

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 operational conditions that may occur for real-world hosted services, we considered three different arrival processes, namely:

- *Deterministic Modulated Poisson Process* (DMPP), to generate workloads exhibiting user behavioral patterns like daily-cycles of activity. In particular, we considered 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) [16] to generate self-similar workloads. In particular, we considered a two-states PMPP, henceforth 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) [17] to generate arrival processes exhibiting temporal burstiness [18]. In particular, we considered 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.

3) Application Manager Configuration: As discussed before, the parameters characterizing the Application Manager are the forgetting factor of the RLS algorithm, the structure of the ARX model, the smoothing factor of the EWMA filter, the weighting matrices of the LQR controller, and the control sampling interval, that were set as follows:

- RLS forgetting factor: we chose the commonly used value of 0.98 which means that the weight of past observations decays very rapidly with time;
- ARX model structure: we found via experiments that an ARX model with 2 poles, 1 zero and a single delay (per input) was good enough to approximate the behavior of our simulated applications.
- EWMA smoothing factor: we used a value of 0.7, so that the influence of past observations does not vanish too fast;
- LQR weighting matrices: we used the *Bryson's rule* [19];
- value of control sampling time: was derived by means of off-line trial-and-error experiments, from which we found that a reasonable value (with respect to control responsiveness and control overhead) was to set it to 8 times the value of the request arrival rate. We used such trial-and-error approach since, to the best of our knowledge, there is no systematic method that can be directly applied to computing systems. The rationale under the choice of using the request arrival rate as a "reference" value for control sampling time is that the arrival rate can be seen as an indicator of how much effort is needed by the controller (i.e., the higher is the arrival rate, the higher is the number of requests the application has to serve, and then the faster the controller needs to operate, and vice versa).

4) Performance Metrics: We assessed the performance of our resource management framework by measuring the number of violation with respect to the SLOs constraints of the various applications, and the amount of electrical energy (expressed in Joule) consumed to serve each submitted service request. For each of these quantities, we computed the 95% confidence level at a relative precision of 4%. This was achieved by using the simulation output analysis method commonly known as *independent replications*, where the terminating condition of each replication was set to be the achievement of at least 1000000 served requests per application. The number of independent replicas needed to achieve, for each simulation experiment, the required confidence level was computed on-line by means of the *relative precision of the confidence interval* method [8].

In our experiments, we used as SLO specification of each application the 0.99^{th} quantile of its response time. This means that the IaaS 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 greater than the SLO value) is greater than 1% during a pre-determined time interval. To compute the SLO value for each application, we used a benchmark-like approach similar to the one described in [3] for application profiling, that consists in running a series of simulations for each application and for each type of arrival process (assigning to each tier exactly the amount of CPU capacity as defined by the reference machine specifications) and measuring the 0.99^{th} quantile of the response time empirical distribution. [2]

B. Results and Discussion

To show the effectiveness of our solution, hereinafter referred to as OUR-SOLUTION, we compared it with two other static approaches, traditionally applied in data center resource management, named in the following as STATIC-SLO and STATIC-ENERGY.

[2] A similar approach has been adopted to compute the operating point used by the on-line system identification algorithm (see Section II-B), such that the operating value for the response time was set to the mean response time observed in each simulation, and the one for the CPU share was set to the ratio between the CPU capacity of the reference machine and the one of the machine used in the simulated system.

In the STATIC-SLO approach, the IaaS provider statically assigns to each VM (running a particular application tier) the amount of CPU capacity defined by the reference machine. This is a SLO-conserving approach since, by assigning exactly the amount of capacity needed to satisfy SLOs, the provider favors SLOs satisfaction in place of energy consumption reduction.

Conversely, in the STATIC-ENERGY approach, the IaaS provider statically assigns to each VM a fixed amount of capacity that is 25% lower than the one defined by the reference machine. This is an energy-conserving approach since, by assigning less capacity than required, the provider favors energy consumption reduction instead of SLOs satisfaction (in the hope to still get a low number of SLO violations).

For the sake of readability, we have grouped the results in four different scenarios, namely S-DMPP, S-PMPP, S-MMPP and S-MIX, which differ from each other in the type of arrival process, whose settings are summarized in Table I (where we used the same notation described in Section III-A). It is important to point out that in each scenario, except for S-MIX, we fixed the type of arrival process and varied the type of application to A_1, A_2, and A_3. Instead, in the S-MIX scenario, we fixed the type of application to A_3 and varied, for each application instance, the type of arrival process. The purpose of this scenario is to show that under different type of time-varying workloads our solution is still effective.

The results of the various scenarios are presented in four separate tables: Table II for S-DMPP, Table III for S-PMPP, Table IV for S-MMPP, and finally Table V for S-MIX. Each row of a given table reports the results obtained, by the various applications, under a specific resource management approach (i.e., STATIC-SLO, STATIC-ENERGY, and OUR-SOLUTION); numbers inside parenthesis (when present) represent the standard deviations of the related measures. The columns of each table have instead the following meaning. The second to fourth columns (each labeled "% SLO violations") show the mean percentage of SLO violations for each application (lower values correspond to better results). The fifth column (labeled "Power Consumption") reports the mean instantaneous power (expressed in Watt) absorbed by the Cloud infrastructure (lower values correspond to better results). The last column (called "Efficiency") is a power-to-performance ratio (in Joule per request) computed as the ratio between the energy consumed by the Cloud infrastructure and the number of satisfied SLOs (averaged over all replications); it represents the mean amount of Joule spent by the Cloud infrastructure in order to satisfy the SLO of a single request (over the whole simulation). We called it "Efficiency" since it provides an indication of how efficiently a given approach used physical resources of the Cloud infrastructure in order to lower the number of SLO violations and, at the same time, to reduce energy consumption; thus, the lower is its value, the better is the result.

By looking at the results reported in the results tables, we can observe that our approach (row OUR-SOLUTION) always achieves a lower number of SLO violations (with respect to the 1% threshold defined by SLO specifications), and always outperforms the STATIC-ENERGY approach. As a matter of fact, in both S-DMPP (see Table II) and S-PMPP (see Table III), the OUR-SOLUTION approach, with respect to the STATIC-ENERGY one, 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 (as instead is for the STATIC-ENERGY approach).

The advantage of the OUR-SOLUTION approach is more evident for both S-MMPP (see Table IV) and S-MIX (see Table V), where the STATIC-ENERGY approach has not been able to converge to the prescribed accuracy (the "n/a" label stands for "result not available") because of the aggregation of too many queueing phenomena (caused by the inability to dynamically adjust CPU shares during high-intensity arrival periods), which resulted in an unstable (simulated) system.

The comparison between OUR-SOLUTION and STATIC-SLO approaches needs more attention. First, the power consumption implied by the OUR-SOLUTION approach is always lower than the one obtained with the STATIC-SLO one, but in S-MIX. Second, both approaches keep the percentage of SLO violations under the 1% threshold (as defined by SLO specifications). However, only for S-DMPP, the percentage of such violations yield by the OUR-SOLUTION approach is always lower than the STATIC-SLO one; in all the other cases, there are mixed situations (e.g., in S-MMPP, the OUR-SOLUTION approach, with respect to the STATIC-SLO one, resulted in a lower percentage of SLO violations for "Application #3" but higher for "Application #1"). This can be ascribed to the following reason. The OUR-SOLUTION approach, unlike the STATIC-SLO approach, is able to dynamically react to high-intensity workload periods by increasing the fraction of resource capacity for the interested application; thus it generally should exhibit a lower percentage of SLO violations than the STATIC-SLO approach. However, there might be cases where such periods overlap in a way that some tier get less resource capacity than needed, thus increasing the probability to violate SLO constraints; indeed, this is the main reason motivating the cases where the percentage of SLO violations obtained with the OUR-SOLUTION approach is greater than the one resulted with the STATIC-SLO one. In any case, it is important to note that the STATIC-SLO approach 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 consequence, this approach implies that the number of VMs that can be consolidated on the same physical machine is lower than those attained by the OUR-SOLUTION (that, instead, allocates to each VM the fraction of CPU capacity it needs). Therefore, the STATIC-SLO approach potentially requires – for a given number of VMs – a larger number of physical resources than the OUR-SOLUTION one, thus yielding a larger energy consumption.

Finally, for the "Efficiency" measure we finds a similar situation: the OUR-SOLUTION approach is always able to

TABLE I
EXPERIMENTAL EVALUATION: SCENARIOS.

Scenario	Application #1			Application #2			Application #3		
	Type	Arrival Process	SLO	Type	Arrival Process	SLO	Type	Arrival Process	SLO
S-DMPP	A_1	DMPP$(1, 5, 10, 3600)$	1.176	A_2	DMPP$(10, 5, 1, 3600)$	0.612	A_3	DMPP$(5, 10, 1, 3600)$	0.608
S-PMPP	A_1	PMPP$(5, 10, 1, 1.5)$	1.245	A_2	Same as Application #1	0.655	A_3	Same as Application #1	0.624
S-MMPP	A_1	MMPP$(5, 15, 0.0002, 0.002)$	4.001	A_2	Same as Application #1	1.962	A_3	Same as Application #1	1.935
S-MIX	A_3	DMPP$(5, 10, 1, 3600)$	0.608	A_3	MMPP$(5, 15, 0.0002, 0.0020)$	1.935	A_3	PMPP$(5, 10, 1, 1.5)$	0.624

TABLE II
EXPERIMENTAL EVALUATION: RESULTS FOR THE S-DMPP SCENARIO.

Approach	Application #1 % SLO violations	Application #2 % SLO violations	Application #3 % SLO violations	Power Consumption Watt	Efficiency Joule/req
STATIC-SLO	0.67% (±0.04%)	0.75% (±0.05%)	0.78% (±0.03%)	1043.59 (±5.27)	61.84
STATIC-ENERGY	19.19% (±0.83%)	14.05% (±0.59%)	19.40% (±0.31%)	1013.04 (±5.29)	72.39
OUR-SOLUTION	0.36% (±0.08%)	0.49% (±0.04%)	0.49% (±0.04%)	1037.69 (±5.78)	61.31

TABLE III
EXPERIMENTAL EVALUATION: RESULTS FOR THE S-PMPP SCENARIO.

Approach	Application #1 % SLO violations	Application #2 % SLO violations	Application #3 % SLO violations	Power Consumption Watt	Efficiency Joule/req
STATIC-SLO	0.86% (±0.21%)	0.78% (±0.11%)	0.69% (±0.22%)	1158.37 (±56.18)	46.16
STATIC-ENERGY	21.91% (±11.32%)	17.41% (±7.31%)	15.58% (±8.72%)	1083.23 (±169.34)	58.44
OUR-SOLUTION	0.88% (±0.41%)	0.75% (±0.31%)	0.59% (±0.32%)	1150.11 (±170.63)	47.34

TABLE IV
EXPERIMENTAL EVALUATION: RESULTS FOR THE S-MMPP SCENARIO.

Approach	Application #1 % SLO violations	Application #2 % SLO violations	Application #3 % SLO violations	Power Consumption Watt	Efficiency Joule/req
STATIC-SLO	0.68% (±0.26%)	0.76% (±0.24%)	0.77% (±0.20%)	1064.90 (±26.07)	60.48
STATIC-ENERGY	n/a	n/a	n/a	n/a	n/a
OUR-SOLUTION	0.81% (±0.28%)	0.77% (±0.24%)	0.66% (±0.18%)	1064.12 (±26.99)	60.38

TABLE V
EXPERIMENTAL EVALUATION: RESULTS FOR THE S-MIX SCENARIO.

Approach	Application #1 % SLO violations	Application #2 % SLO violations	Application #3 % SLO violations	Power Consumption Watt	Efficiency Joule/req
STATIC-SLO	0.77% (±0.05%)	0.77% (±0.13%)	0.53% (±0.10%)	1029.71 (±29.38)	52.25
STATIC-ENERGY	n/a	n/a	n/a	n/a	n/a
OUR-SOLUTION	0.77% (±0.05%)	0.76% (±0.22%)	0.64% (±0.19%)	1036.31 (±45.92)	50.87

use physical resources more efficiently than the STATIC-SLO approach, with the exception of the S-PMPP scenario where, however, the power consumption of the OUR-SOLUTION approach is lower than the value attained by the STATIC-SLO one.

In conclusion, the OUR-SOLUTION approach is able to achieve better results compared to other approaches both in terms of efficiency and energy consumption.

IV. RELATED WORK

In this section, we provide a brief state-of-the-art about dynamic performance- and power-aware resource management in Cloud infrastructures. Over last years, many research works have arisen for dynamically managing physical resources of a Cloud infrastructure in order to take into consideration performance targets of hosted applications and power consumption of the infrastructure. However, the majority of them dealt with the two aspects separately (e.g., see [20]–[22] and [23]–[25] for disjoint performance-aware and power-aware resource management approaches, respectively). Only very recently, several works, combining dynamic power- and performance-aware resource management, have published in the literature. For instance, in [26], a combined predictive and reactive provisioning technique is proposed, where the predictive component estimates the needed fraction of capacity according to historical workload traces 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 on-line system identification, is potentially able to work with any type of workload without any prior knowledge.

In [27] a two-levels control architecture for coordinating power and performance management in virtualized clusters is proposed, where, unlike our framework, the achievement of performance targets is always subjected to the reduction of power consumption; conversely, in our framework, the reduction of power consumption is constrained to the achievement of performance targets.

In [28], a two-layers hierarchical control approach is used for tackling the resource provisioning problem for multi-tier Web applications. Even though this work share some architectural similarity with our framework, there are two important differences; the first difference regards the way ARX parameters are computed (i.e., through off-line identification), while the second one is about 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 physical machine).

In [29], a two-level resource management is proposed which combines a utility-based approach to constraint programming; it differs from our work in that the architecture of the framework is not decentralized since both the utility maximization and the constraints satisfaction problems need a global view of the computing system state.

Finally, in [30] a time-hierarchical and decentralized framework, able to dynamically place VMs and to turn on or off physical machines when needed, is presented. Besides of the conceptual similarity with our framework (including the Migration Manager component), this work differs from our one for the approach taken to minimize power consumption and SLO violations, which consists in a utility-based approach combined to mixed integer non-linear programming. [3]

V. Conclusions and Future Work

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. By means of simulation, we showed that, compared to traditional static approaches, our work is able to dynamically adapt to time-varying workloads and, at the same time, to significantly reduce both SLO violations and energy consumption.

Regarding possible future works, there are some interesting activity that can be carried on. Future work includes the extensions of the proposed framework with two new components, namely the Migration Manager discussed in Section II and the Resource Negotiator (to negotiate physical resources among different Cloud infrastructures), and its integration into a real testbed.

References

[1] P. Mell et al, "Cloud Computing: Recommendations of the National Institute of Standards and Technology," NIST, Spec. Pub. 800-145, Jan 2011.

[3] It is important to note that, since we have not implemented the Migration Manager yet, we are unable to perform a truly comparison, which thus is postponed to a future work, as discussed in the next section.

[2] B. Pring et al, "Forecast: Sizing the Cloud; understanding the opportunities in cloud services," Gartner, Inc., Research Report G00166525, Mar 2009.

[3] T. Wood et al, "Profiling and modeling resource usage of virtualized applications," in *Proc. of the* 9^{th} *ACM/IFIP/USENIX Int. Conf. on Middleware (Middleware'08)*, 2008, pp. 366–387.

[4] V. Kumar et al, "A state-space approach to SLA based management," in *Proc. of the* 11^{th} *IEEE/IFIP Network Operations and Management Symposium (NOMS'08)*, 2008, pp. 192 –199.

[5] L. Mieritz et al, "Defining Gartner total cost of ownership," Gartner, Inc., Research Report G00131837, Dec 2005.

[6] ENERGY STAR Program, "Report to congress on server and data center energy efficiency," U.S. EPA, Tech. Rep., Aug 2007.

[7] W. Vogels, "Beyond server consolidation," *Queue*, vol. 6, no. 1, pp. 20–26, 2008.

[8] J. Banks et al, *Discrete-Event System Simulation*, 5th ed. Prentice Hall, 2010.

[9] K. J. Åström et al, *Adaptive Control*, 2nd ed. Addison-Wesley, 1994.

[10] L. Ljung, *System Identification: Theory for the User*, 2nd ed. Prentice Hall, 1999.

[11] J. P. Hesphana, *Linear Systems Theory*. Princeton Press, 2009.

[12] S. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 1, no. 3, pp. 239–250, 1959.

[13] H. Kwakernaak et al, *Linear Optimal Control Systems*. Wiley-Interscience, 1972.

[14] X. Fan et al, "Power provisioning for a warehouse-sized computer," in *Proc. of the* 34^{th} *Int. Symp. on Computer Architecture (ISCA'07)*, 2007, pp. 13–23.

[15] "SPECpower_ssj2008 benchmark." [Online]. Available: http://www.spec.org/power_ssj2008

[16] T. Le-Ngoc et al, "A Pareto-modulated Poisson process (PMPP) model for long-range dependent traffic," *Comput. Comm.*, vol. 23, no. 2, pp. 123–132, 2000.

[17] W. Fischer et al, "The Markov-modulated Poisson Process (MMPP) cookbook," *Perform. Eval.*, vol. 18, no. 2, pp. 149–171, 1993.

[18] N. Mi et al, "Injecting realistic burstiness to a traditional client-server benchmark," in *Proc. of the* 6^{th} *IEEE Int. Conf. on Autonomic Computing (ICAC'09)*, 2009, pp. 149–158.

[19] A. E. Bryson, Jr. et al, *Applied Optimal Control: Optimization, Estimation, and Control*, revised ed. Taylor & Francis, 1975.

[20] B. Urgaonkar et al, "Analytic modeling of multitier internet applications," *ACM Trans. Web*, vol. 1, no. 1, 2007.

[21] P. Padala et al, "Automated control of multiple virtualized resources," in *Proc. of the* 4^{th} *ACM European Conf. on Computer Systems (EuroSys'09)*, 2009, pp. 13–26.

[22] P. Bodík et al, "Statistical machine learning makes automatic control practical for Internet datacenters," in *Proc. of the 2009 USENIX Conf. on Hot Topics in Cloud Computing (HotCloud'09)*, 2009.

[23] R. Nathuji et al, "VirtualPower: Coordinated power management in virtualized enterprise systems," in *Proc. of* 21^{st} *ACM SIGOPS Symp. on Operating Systems Principles (SOSP'07)*, 2007, pp. 265–278.

[24] R. Raghavendra et al, "No "power" struggles: Coordinated multi-level power management for the data center," in *Proc. of the* 13^{th} *Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIII)*, 2008, pp. 48–59.

[25] A. Verma et al, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Proc. of the* 9^{th} *ACM/IFIP/USENIX Int. Conf. on Middleware (Middleware'08)*, 2008, pp. 243–264.

[26] A. Gandhi et al, "Minimizing data center SLA violations and power consumption via hybrid resource provisioning," in *Proc. of the* 2^{nd} *Int. Green Computing Conference (IGCC'10)*, 2010.

[27] X. Wang et al, "Coordinating power control and performance management for virtualized server clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 2, pp. 245–259, 2011.

[28] P. Xiong et al, "Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach," in *Proc. of the* 31^{st} *Int. Conf. on Distributed Computing Systems (ICDCS'11)*, 2011, pp. 571–580.

[29] H. N. Van et al, "Performance and power management for cloud infrastructures," in *Proc. of the 2010 IEEE* 3^{rd} *Int. Conf. on Cloud Computing (CLOUD'10)*, 2010, pp. 329–336.

[30] D. Ardagna et al, "Energy-aware autonomic resource allocation in multitier virtualized environments," *IEEE Trans. on Services Comput.*, vol. 99, no. PrePrints, 2010.