A Hierarchical Receding Horizon Algorithm for QoS-driven control of Multi-IaaS Applications

Danilo Ardagna, Michele Ciavotta, Riccardo Lancellotti, Michele Guerriero

Abstract—Cloud Computing is emerging as a major trend in ICT industry. However, as with any new technology, new major challenges lie ahead, one of them concerning the resource provisioning. Indeed, modern Cloud applications deal with a dynamic context that requires a continuous adaptation process in order to meet satisfactory Quality of Service (QoS) but even the most titled Cloud platform provide just simple rule-based tools; the rudimentary autoscaling mechanisms that can be carried out may be unsuitable in many situations as they do not prevent SLA violations, but only react to them. In addition, these approaches are inherently static and cannot catch the dynamic behavior of the application and are unsuitable to manage multi-Cloud/data center deployments required for mission critical services. This situation calls for advanced solutions designed to provide Cloud resources in a predictive and dynamic way. This work presents capacity allocation algorithms, whose goal is to minimize the total execution cost, while satisfying some constraints on the average response time of multi-Cloud based applications. This paper proposes a joint load balancing and receding horizon capacity allocation techniques, which can be employed to handle multiple classes of requests. An extensive evaluation of the proposed solution against an Oracle with perfect knowledge of the future and well-known heuristics proposed in the literature is provided. The analysis shows that our solution outperforms the heuristics producing results very close to the optimal ones, and reducing the number of QoS violations (in the worst case QoS constraints violation rate is 4.259% versus up to 17.245% of other approaches and can easily reduced by roughly a factor of 4 by exploiting the receding horizon approach). Furthermore, a sensitivity analysis over two different time scales indicates that finer grained time scales are more appropriate for spiky workloads, whereas smooth traffic conditions are better handled by coarser grained time scales. Analytical results are validated through simulation, which also analyzes the impact of Cloud environment random perturbations. Finally, experiments on a prototype environment demonstrate the effectiveness of the proposed approach under real workloads.

Index Terms—Auto-Scaling, Multi-Cloud, Capacity Allocation, Load Sharing, Optimization, QoS

1 Introduction

Cloud computing has been a major driving force for the evolution of the Information and Communication Technology (ICT) industry over the last years. The main ICT players (e.g., Google [1], Amazon [2], and Microsoft [3]) have constantly improved cost-effectiveness, reliability as well as the overall computing power consumption of Cloud systems. This effort has made the Cloud a tool mature and suitable for business (outside ICT): it is no longer surprising to see companies operating in very different fields shifting their business models in order to benefit from the advantages associated with this paradigm, which are mainly due to high elasticity, scalability and cost savings [4].

Nonetheless, Cloud computing, despite the massive popularity gained over the years, still entails several challenges, especially in the area of resource provisioning. In particular, it has emerged evident the need for Cloud providers to guarantee adequate levels of Quality of Service (QoS) to their customers [5]. For this to be achieved, advanced operations solutions are needed to provide support to performance prediction, monitoring of Service Level Agreements (SLAs), and adaptive configuration while satisfying requirements on cost-effectiveness, reliability, and security. Current solutions for enforcing SLAs mainly address single Clouds [6], [7], [8]; however, providing SLA guarantees with a multi-Cloud scope is fundamental when availability is a major requirement (e.g., mission critical services). Still, when a multi-Cloud scenario is contemplated, in which several applications are running simultaneously on data centers belonging to different providers or, alternatively, on separate data centers belonging to the same provider, only basic solutions are available. These usually implement a purely reactive approach (e.g., the Amazon autoscaling rules [9]) where scaling actions are blindly triggered when a threshold on some monitoring metric is exceeded (e.g., CPU utilization is above 60%).

In an attempt to bridge this gap, a novel resource allocation solution that is able to dynamically adapt the Cloud resources in order to satisfy SLAs and to minimize cloud usage costs is proposed facing the wider scenario where different applications are hosted across multiple geographically distributed Clouds. The proposed approach aims at unleashing the full potential of the Cloud paving the way to a class of services featuring mission-critical availability at minimum cost. The resource allocation scheme operates at two levels, optimally balancing the incoming requests at inter-cloud level while optimally allocating the computational capacity at intra-cloud one. More in details, this paper proposes a dual time scale approach working on different scales, both temporal and spatial (single vs. multi-Cloud). At a coarse-grained time scale (e.g., about one hour), an innovative technique for the distribution of requests among multiple Cloud providers is proposed. At the fine-grained time scale (e.g., every 5 or 10 minutes) a receding horizon algorithm allocates virtual machines (VMs) in each data center meeting QoS constraints (extending [10]). In both cases, the algorithms exploit predictive models for workload forecasting. Both the long- and short-term

- *D. Ardagna and M. Guerriero are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy. Email: {name.lastname}@polimi.it*
- *Michele Ciavotta is with the Dipartimento di Informatica, Sistemistica e Comunicazione, Universitá di Milano-Bicocca, Milano, Italy. Email: michele.ciavotta@unimib.it*
- *Riccardo Lancellotti is with the Dipartimento di Ingegneria "Enzo Ferrari", Universitá di Modena e Reggio Emilia, Modena, Italy. Email: riccardo.lancellotti@unimore.it*

problems are formulated as mixed integer linear programming (MILP) problems.

The contribution of this paper is not only limited to the algorithmic proposal but validates the resource allocation techniques demonstrating that the proposed solutions save costs without incurring significant SLA violations through three types of scenarios. First, analytical models are used to compare the proposed solution with an *Oracle* with perfect knowledge about the future [11] and with well-known heuristics proposed in the literature [12], [13], [14] based on utilization thresholds. Second, a new simulator that captures the data center behavior including the presence of exogenous effect on the overall performance, modeled through *random environments* [15], has been developed to analyze the behavior of the resource allocation solutions under a number of scenarios of interest (e.g., cloud resource contention, queue length effect, workload spikes). In this way, the simulator allows evaluating the SLA violations in realistic situations. Furthermore, the simulator is used for a thorough sensibility analysis with respect to the time granularity of the control strategy, the random environments parameters and the workload scenarios. Third, the results achieved on a prototype environment, developed within the scope of the MODAClouds project [16], [17] are also presented, which demonstrate the viability of the proposed algorithms.

Results confirm that our solution outperforms the heuristics based on utilization thresholds producing results very close to the optimal ones, which can be achieved by the Oracle (cost savings range is [30, 80]%) and reducing the number of SLA violations (in the worst case QoS constraints violation rate is 4.259% versus up to 17.245% of other approaches and can easily reduced by roughly a factor of 4 by exploiting the receding horizon approach). A scalability analysis also demonstrates that both the long- and short-term resource allocation problems can be solved in less than two minutes even for large size instances. Therefore, the solutions can be computed according to the time granularity (1 hour and 5-10 minutes, respectively) which characterizes the problems themselves leaving to the Cloud infrastructure enough time to actuate the changes in the system configuration (e.g., to start additional VMs before the next control time horizon). Moreover, results demonstrated that the solution time grows almost linearly with the problem instance size.

Furthermore, a sensitivity analysis over the two time scales indicates that finer grained time scales are more appropriate for spiky workloads, whereas smooth traffic conditions are better handled by coarser grained time scales. The simulation shows that the overall solution is robust to Cloud environment random perturbations. Finally, the results achieved in the prototype environment demonstrated that the percentage of SLA violations in a Cloud deployment under a real workload is less than 2%.

In our previous contribution [10], we presented the early receding horizon approach used for the fine-grained time scale short-term problem. In this paper we extend on our previous work by i) integrating the receding horizon technique with a long-term and coarse-grain workload management strategy, ii) extending the model by embracing a multi-Cloud vision, iii) providing a more in-depth validation that includes also experiments on a real Cloud deployment.

The remainder of this paper is organized as follows: In Section 2 the global problem of managing the delivery of web-based services over multiple Cloud infrastructures is presented and discussed. Section 3 provides the mathematical programming formulation of the short- and long-grained resource allocation problems. Section 4 describes the algorithms used for both the request distribution over multiple data centers and for the VM allocation at the level of a single data center. Section 5 evaluates the quality of our solution through experiments and simulation. In Section 6 other literature approaches are reviewed. Finally, Section 7 reports concluding remarks.

2 Problem Statement and Assumptions

This section aims at introducing the design assumptions and system models used as a reference throughout the paper. In particular, hierarchical modeling of the problem is proposed working on the two different temporal and scope granularities, namely long-term (introduced in Section 2.3) and short-term (discussed in Section 2.2).

2.1 Problem overview

This paper strives to capture the perspective of a Software as a Service (SaaS) provider that runs a suite of applications, in form of Web Services (WS), across multiple Infrastructure-as-a-Service (IaaS) providers. Such applications are heterogeneous in terms of resource demands, workload profiles they are subject to, and SLA requirements.

Figure 1 depicts the reference multi-Cloud environment considered in this paper. The system serves a set $\mathcal{K}$ of WS classes, where each class corresponds to a specific web application (hereinafter the terms *WS application* and *request class* will be used interchangeably). For each class $k \in \mathcal{K}$, the incoming workload arrival rate is denoted by Λ_k (expressed in terms of number of requests/s). Applications are deployed on virtual machines (VMs), which are instantiated on-demand and on a pay-per-use basis by a set $\mathcal{I}$ of IaaS providers. For the sake of simplicity, this work assumes that a VM cannot be shared, that is it can only host a single WS application (this hypothesis can be easily relaxed). Services can be replicated across multiple Clouds or, equivalently, across different data centers of the same provider (e.g., regions or availability zones of the Amazon EC2 platform [18]).

Let us assume that, within the scope of a single Cloud, all the VMs instances are homogeneous in terms of their computing capacity C_i, share the incoming workload evenly, and their performance are never subject to contention issues. This is a legitimate assumption as it corresponds, within a reasonable range, to the solution currently implemented by IaaS providers (e.g., [19]). As far as costs are concerned, two pricing models are offered by every provider: they supply *reserved* and *on-demand* VMs. Let δ_i and ρ_i denote the hourly fees for on-demand and reserved instances ($\rho_i < \delta_i$), respectively, whereas W_i indicates the overall number of available reserved instances. As for the service rate, let μ_k represent as the maximum service rate (measured in requests/s) featured by a VM of unitary capacity hosting the WS application k.

Furthermore, let us assume that an SLA contract associated with each WS class $k \in \mathcal{K}$ is established between the SaaS provider and her/his customers. This contract specifies the QoS levels expressed as a bound on the average response time R_k for each class k. More in details, given a threshold $\overline{R}_k$ on the average response time, the SLA in force for WS application k can be expressed as $R_k \leq \overline{R}_k$. However, to avoid large disparities in response times across customer classes (as a result of sticky sessions the flow of requests from certain customers could be consistently redirected to the Cloud with the poorest or best

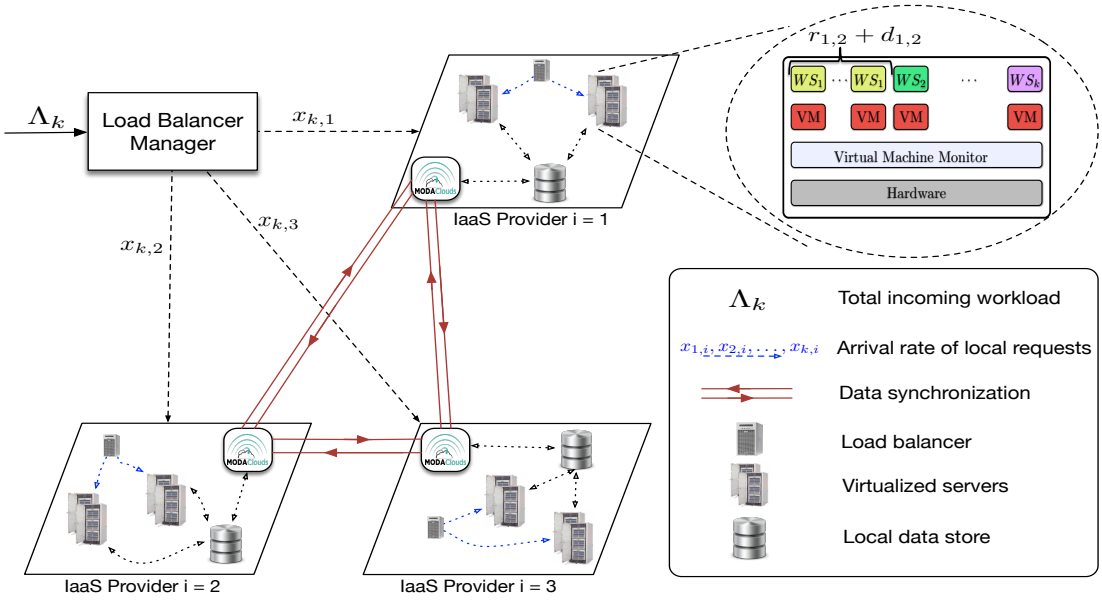

Fig. 1. Cloud infrastructures and data migration and synchronization system.

performance [20]), a reinforced constraint balancing mechanism was introduced requiring that $R_{k,i} \leq \overline{R}_k$ for each provider i.

In this work, the problem of minimizing SaaS operations costs is modeled as a joint multi-Cloud Capacity Allocation (CA) and Load Sharing (LS) problem and efficiently solved by means of a hierarchical optimization approach that considers the two different long and short-term time scales.

At long-term, the inter-cloud load sharing problem is addressed; at this level, a decision is made on how the next-hour total expected workload is allocated to different IaaS providers with the objective of minimizing the compound VM leasing costs. In other words, every hour the long-term algorithm splits, for each application k, the prediction for the global incoming request flows $\tilde{\Lambda}_k$[1] into the request flows for each IaaS provider $x_{k,i}$ $k \in \mathcal{K}, i \in \mathcal{I}$ taking into account the costs for allocating VMs on the different providers. This is a long-term problem because the algorithm is executed on about an hourly basis to avoid control instability issues; moreover, when it comes to reacting sudden spikes in the workload one-hour horizon is considered a very long time [21], [22]. The long-term problem time horizon will be denoted by T_{long} that represents the number of hours for which the workload allocation is calculated. It is worth noticing that in order to calculate the workload shares, optimized with respect to the allocation costs, the long-term algorithm solves both LS and CA problems. More details can be found in Sections 2.2 and 3.1.

The short-term problem, in turn, deals with the intra-Cloud CA optimization, where the time-scale is in the order of 5 or 10 minutes, which considers as incoming workload the load distribution share $x_{k,i}$ resulting from the long-term problem. It operates on a time horizon denoted by T_{short} (one hour), as well as on time slots of duration T_{slot} The objective is to determine, for each Cloud i, the minimum-cost number of VM to serve the assigned percentage of the incoming workload while guaranteeing the average response time is below a given threshold $\overline{R}_k$, i.e., $R_{k,i} \leq \overline{R}_k$. More details are available in Sections 2.3 and 3.2.

The problem considered in this work entails two types of decisions, with a strong liaison with phenomena observable in Web applications deployed in the Cloud. The decision made at the higher level is of *tactical* nature as it concerns the load sharing over a long time period. This approach is legitimized by the characteristics of the incoming workload when observed at a coarse-grain scale (such as for example one hour). This is generally smoothed and free of instantaneous peaks, and can, therefore, be predicted with high accuracy and reliability. This guarantee the validity of the decision made also for shorter time periods with the undoubted advantage of simplifying the overall decision-making process. At finer-grain, however, workload traces generally exhibit an erratic behavior characterized by high variability due to the short-term changes of the typical Web-based workload. For this reason, shorter time scales are used to made decisions of operational and control nature as the allocation of new VM instances with the purpose of avoiding saturation conditions and exceeding the thresholds in QoS constraints. Concerning the workload prediction, several methods have been adopted over the last decades [21], [23] (e.g., ARMA models, exponential smoothing, and polynomial interpolation), making them suitable to forecast seasonal workloads, common at coarse time scales (e.g., day or hour), as well as runtime and non-stationary request arrivals characterizing the time scales (few minutes) considered here. In general, each prediction mechanism is characterized by several alternative implementations, where the choice about filtering or not filtering input data (usually a runtime measure of the metric to be predicted) and choosing the best model parameters in a static or dynamic way are the most significant. However, workload prediction is out of the scope of this paper.

To conclude, it is worth remarking that the presented multi-Cloud scenario is technologically feasible thanks to, for instance, the software solutions developed within MODAClouds project [16], [17], which encompass full-stack modeling, deploy-

1. In the following $\tilde{}$ is used to denote the prediction of a given parameter. For example, Λ_k denotes the real incoming workload while $\tilde{\Lambda}_k$ denotes its prediction.

ment, and run time management of multi-Cloud applications, removing technological limitations and lock-ins that have so far prevented the full potential of this approach (high availability and cost saving due to IaaS competition and workload redistribution at run time [24]) from being exploited. In particular, since one of the main inhibitors to the multi-Cloud adoption is the objective technological challenge of keeping updated the geographically distributed and technologically distinct databases, the above-mentioned project provides a distributed middleware (see Figure 1) in charge of reliably synchronizing data among (even technologically) different databases [25].

2.2 Long-term request distribution mechanism

The long-term problem concerns the distribution of the incoming requests load $\Lambda_k, k \in \mathcal{K}$ to the providers of the set $\mathcal{I}$, with a time scale in the order of one hour, minimizing the cost for the instances allocated. The problem is solved one hour in advance based on the workload prediction; in this way, the outcome of the long-term problem can be used to set up the inter-Cloud Load Balancer Manager (see Figure 1), which distributes the load for the short-term problem (one for each provider).

In the resolution of the long-term problem, the decision variable considered are the number of reserved and on-demand VM instances for each provider and for each class, denoted $r_{k,i}$, and $d_{k,i}$, respectively, as well as the request rate forwarded to each Cloud provider $x_{k,i}$. A performance model based on queueing theory is used in order to evaluate the average response time ($R_{k,i}$) given the number of VMs supporting a WS application ($r_{k,i} + d_{k,i}$), and the incoming workload ($x_{k,i}$). More in details, each WS application hosted on a VM is modeled as an M/G/1 queue in tandem with a delay center as in [26]. As delineated in Figure 2 the model features multiple servers (i.e., VMs) running in parallel to support the same WS application. The incoming requests are assumed to be served according to the processor sharing scheduling policy, which is frequently used by WS containers [27]. M/G/1 models are widely used in the literature to estimate WS applications performance [28], [29]. The delay center $D_{k,i}$ is, in its most general definition, an application and Cloud-dependent parameter used to model network and protocol delays introduced in establishing connections, indirect routing, etc.

Although system performance metrics have been evaluated by adopting alternative analytical models (e.g., [28], [30]) and accurate performance models exist in the literature for WS systems (e.g., [31]), there is a fundamental trade-off between the accuracy of the models and the time required to estimate system performance metrics. Given the strict time constraints for evaluating performance metrics in the present setting (especially for the short-term problem), the high complexity of analyzing even small-scale instances of existing performance models has prevented us from exploiting such results here. It is important to note, however, that highly accurate approximations for general queueing networks [32], having functional forms similar to M/G/1 formula, can be directly incorporated into the optimization frameworks.

2.3 Short-term receding horizon control mechanism

The short-term problem is managed at the local level of a single Cloud provider (intra-Cloud). The data center performance is modeled by means of the same queue system used for the long-term problem (see Figure 2). Moreover, let us assume that the performance parameters are continuously updated at run time

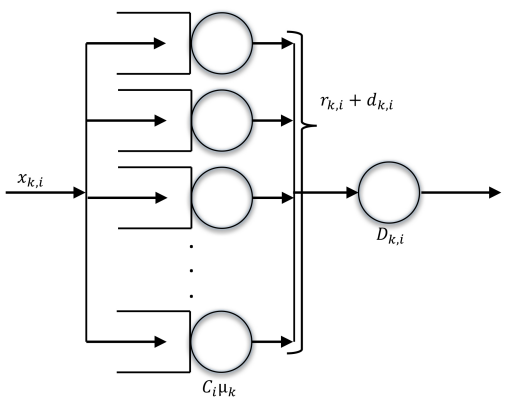

Fig. 2. System performance model

(see [26] for further details) in order to capture transient behaviors, VMs network, I/O interference [33], and time-dependent performance of Cloud resources [34], [35]. To model the time in the short-term problem, multiple time slots of duration T_{slot} are considered; furthermore, within the time horizon T_{Short} (one hour), the problem considers a sliding window of $\mathcal{T}_w$ future time slots. Since the problem deals with time scales finer than one hour, which is considered the minimum instance charging interval T_c, it assumes to pay for an instance as soon as it is required. In other words, the problem formulation assumes that a VM becomes immediately available and it is deemed to be freely accessible until the end of its lease term. One hour later, if the instance is no longer needed it will be released; otherwise, the fee will be charged again and the VM will remain available one more hour. The overall number of time slots in the lease term (i.e., one hour) is denoted by n_c and it is assumed to be integer for simplicity. In the problem formulation, the number of instances already paid and available in each time slot of the time horizon is represented by means of parameters $(\overline{r}^1_{k,i}, \ldots, \overline{r}^{n_c}_{k,i})$ and $(\overline{d}^t_{k,i}, \ldots, \overline{d}^{n_c}_{k,i})$ for reserved and on-demand instances, respectively.

The decision variables of the short-term problem are $(r^1_{k,i}, \ldots, r^{n_w}_{k,i})$ and $(d^1_{k,i}, \ldots, d^{n_w}_{k,i})$, i.e., the number of reserved and on-demand VMs to be started up during the observation window $\mathcal{T}_w = \{1, \ldots, n_w\}$ that, in conjunction with available instances of both types (that is those for which the lease has not yet finalized), have to serve the predicted incoming workload $(\tilde{x}^1_{k,i}, \ldots, \tilde{x}^{n_w}_{k,i})$. The final goal is to minimize the aggregate leasing costs to serve the predicted arrival rate while guaranteeing that the average response time of each WS application is lower than the SLA threshold.

The short-term solution algorithm follows the receding horizon control principle [36], where the optimal solution achieved considering the whole time window, yet the algorithm enforces only the decisions calculated for the nearest time slot. This means that the values $(r^t_{k,i}, d^t_{k,i})$ are calculated for every future time interval of $\mathcal{T}_w$, but the algorithm acts on the controlled system by starting up the optimal number VMs calculated for the first time slot, that is $(r^1_{k,i}, d^1_{k,i})$. The short-term optimization process is then repeated sliding the time window and considering the second time slot as the new starting point. $\mathcal{T}_c$ denotes the set of slots within the VM lease term. Figure 3 graphically illustrates the relationships between $\mathcal{T}_c$, $\mathcal{T}_w$ and T_{slot}. In this work, time slots of 5 and 10 minutes are considered as well as observation windows with n_w from 3 up to 5 time slots, that is ranging from 15 to 50 minutes. As

regarding the charging interval $\mathcal{T}_c$, having considered the common VM lease term of one hour, it resulted composed of 6 or 12 time slots.

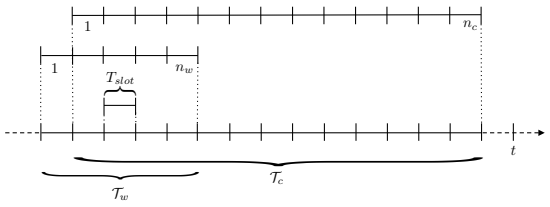

Fig. 3. Relationships between $\mathcal{T}_c$, $\mathcal{T}_w$ and T_{slot} over time

For sake of clarity, the notation adopted in this paper is summarized in Tables 1 and 2.

3 Optimization problems formulation

This section provides the mathematical formulation of the long-term (Section 3.1) and short-term (Section 3.2) optimization problems.

3.1 Long-term problem

The crux of the long-term problem is to split the workload prediction for application k (denoted by $\tilde{\Lambda}_k$) among its replicas hosted on different Clouds with the objective of minimizing the resource leasing costs. The problem is solved hourly by means of a mathematical model, with the scope limited to the following hour. In what remains of this section, the main principles and relationships underlying the model will be introduced briefly, then the mathematical model associated with the problem will be presented and discussed in detail.

From the system performance model presented in Section 3.1 (i.e., M/G/1) the average response time for class k at provider i can be derived in closed form as:

$$R_{k,i} = \frac{1}{C_i\mu_k - \frac{x_{k,i}}{r_{k,i}+d_{k,i}}} + D_{k,i} \tag{1}$$

Moreover, enforcing the M/G/1 equilibrium condition, which avoids infinite queue lenght at VMs [29], [32] it holds:

$$x_{k,i} < C_i\mu_k \left(r_{k,i} + d_{k,i}\right)$$

that can be rearranged as:

$$r_{k,i} + d_{k,i} > \frac{x_{k,i}}{C_i\mu_k}$$

Consequently, the average response time, $R_{k,i}$, can be expressed according to the formula:

$$R_{k,i} = \frac{r_{k,i} + d_{k,i}}{C_i\mu_k \left(r_{k,i} + d_{k,i}\right) - x_{k,i}} + D_{k,i}$$

which, by imposing the QoS condition $R_{k,i} \leq \overline{R}_k$ leads to the following inequality after some algebra:

$$r_{k,i} + d_{k,i} \leq \left(\overline{R}_k - D_{k,i}\right)\left[C_i\mu_k \left(r_{k,i} + d_{k,i}\right) - x_{k,i}\right]$$

and then:

$$r_{k,i} + d_{k,i} + x_{k,i}\left(\frac{\overline{R}_k - D_{k,i}}{1 - \left(\overline{R}_k - D_{k,i}\right)C_i\mu_k}\right) \geq 0$$

TABLE 1
Parameters of the Capacity Allocation Problem.

Global parameters	
$\mathcal{I}$	Set of IaaS providers
C_i	VMs instance capacity of provider i
δ_i	Time unit cost (measured in dollars) for *on-demand* VMs of provider i
ρ_i	Time unit cost (measured in dollars) for *reserved* VMs of provider i
$\mathcal{K}$	Set of WS classes
$D_{k,i}$	Queueing delay (measured in s) for processing WS class k requests at provider i
$R_{k,i}$	Average response time (measured in s) for WS class k request at provider i
$\overline{R}_k$	Average response time threshold (measured in s) for WS class k request
W_i	Maximum number of *reserved* instances available, at provider i
μ_k	Maximum service rate (measured in requests/s) of a capacity 1 VM for executing WS class k requests
Long Term Problem	
$\mathcal{T}_{long}$	Long-term CA time horizon, measured in hours
$\tilde{\Lambda}_k$	Prediction for the total exogenous arrival rate (measured in requests/sec) for WS class k for the whole Cloud system
γ_i	Minimum percentage of traffic distributed to each provider i
Short Term Problem	
$\mathcal{T}_w$	duration of the window of observation
$\mathcal{T}_c$	duration of the charging interval
T_{slot}	Short-term CA time slot, measured in minutes
n_c	Number of time slots within the charging interval $\mathcal{T}_c$
n_w	Number of time slots within the time window $\mathcal{T}_w$
$\overline{r}^t_{k,i}$	Number of *reserved* VMs available for free for the time slot t in the interval under analysis, for class k requests, at provider i
$\overline{d}^t_{k,i}$	Number of *on-demand* VMs available for free for the time slot t in the interval under analysis, for class k requests, at provider i
$x^j_{k,i}$	Real local arrival rate (measured in requests/s) for WS class k, at provider i and at time slot j
$\tilde{x}^t_{k,i}$	Local arrival rate prediction (measured in requests/s) for WS class k, at provider i and at time slot t

TABLE 2
Decision variables of the Capacity Allocation Problem.

Long Term Problem	
$d_{k,i}$	Number of *on-demand* VMs to be allocated for WS class k request, at provider i
$r_{k,i}$	Number of *reserved* VMs to be allocated for WS class k request, at provider i
$x_{k,i}$	Arrival rate (measured in requests/s) allocated to provider i, for class k request
Short Term Problem	
$d^t_{k,i}$	Number of *on-demand* VMs to be allocated for WS class k request at time slot t at provider i
$r^t_{k,i}$	Number of *reserved* VMs to be allocated for WS class k request at time slot t at provider i

Note that it can be safely presumed that $1 - \left(\overline{R}_{k,i} - D_{k,i}\right)C_i\mu_k < 0$ since $C_i > 0$, $\mu_k > 0$, whereas $\overline{R}_{k,i} \gg D_{k,i}$ and $\overline{R}_k \gg \frac{1}{C_i\mu_k}$ are well-accepted assumptions (i.e., the QoS threshold has to be higher than the queueing network delay D_k and the request service time $\frac{1}{C_i\mu_k}$).

Hence, being δ_i and ρ_i the costs of *on-demand* and *reserved* VM instances for provider i, respectively, the joint Capacity Allocation and Load Sharing problem can be formulated as:

$$(P_{lt}) \qquad \min_{r_{k,i}, d_{k,i}, x_{k,i}} \sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}} \left(\rho_i r_{k,i} + \delta_i d_{k,i}\right) \tag{2}$$

Subject to the conditions:

$$r_{k,i} + d_{k,i} - \frac{x_{k,i}}{C_i \mu_k} > 0 \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K}, \quad (3)$$

$$r_{k,i} + d_{k,i} + \frac{x_{k,i}(\overline{R}_k - D_{k,i})}{1 - (\overline{R}_k - D_{k,i})\, C_i \mu_k} \geq 0 \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K}, \quad (4)$$

$$\sum_{i \in \mathcal{I}} x_{k,i} = \tilde{\Lambda}_k \qquad \forall k \in \mathcal{K}, \quad (5)$$

$$x_{k,i} \geq \gamma_i \tilde{\Lambda}_k \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K}, \quad (6)$$

$$\sum_{k \in \mathcal{K}} r_{k,i} \leq W_i \qquad \forall i \in \mathcal{I}, \quad (7)$$

$$r_{k,i} \geq 0,\ r_{k,i} \in \mathbb{N} \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K}, \quad (8)$$

$$d_{k,i} \geq 0,\ d_{k,i} \in \mathbb{N} \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K}. \quad (9)$$

(P_{lt}) can be classified as a Mixed Integer Linear Programming (MILP) problem, since the variables of the problem are integer or float and the objective function and constraints are linear. In particular, it minimizes the total leasing costs considering both *on-demand* $r_{k,i}$ and *reserved* $d_{k,i}$ over a time-period of one hour; to achieve this goal the model has to decide not only the number of instances but also the optimal load sharing policy, $x_{k,i}$ (which does not contribute directly to the value of the objective function, though). As a result of the problem, the workload of the system for the next hour is redirected to each provider i and application k according to the probability defined by $\frac{x_{k,i}}{\sum_{i' \in \mathcal{I}} x_{k,i'}}$.

As far as the constraints are concerned, inequalities (3), as discussed earlier, impose the equilibrium distribution to the underlying M/G/1 queue models (one per application k and provider i). This constraint set strongly relates to (4), which implements the response time bounds defined in the service level agreements. The equilibrium conditions (3), in fact, imply the positivity of the denominator in the response time formula (1).

Constraints (5) (6) define how the model can split the predicted workload; on the one hand, constraints (5) ensure that the traffic assigned to individual providers equals the overall load predicted for application k. Thus, the model is forced to assign the whole traffic. On the other hand, constraints (6) guarantee that every IaaS receives at least a fraction γ_i of workload, preventing scenarios where all the workload is simply forwarded to the minimum-cost provider featuring the most economically advantageous offer.

The constraint set (7), creates of a liaison between different classes of applications making the problem much harder to solve: at the same provider, in fact, the model is allowed to allocate the maximum number W_i of *reserved* instances, considering all whole set of applications deployed; without this constraint, the problem would be separable in $|\mathcal{K}|$ independent sub-problems.

3.2 Short-term problem

The short-term problem is addressed starting with the solution of the long-term one. However, the time granularity of the problem is much finer, thus motivating the differences in the problem model. The Capacity Allocation (CA) problem is solved over an observation window $\mathcal{T}_w$ of n_w time slots and aims at minimizing the overall costs for *reserved* and *on-demand* instances to serve the predicted arrival rate $\tilde{x}^t_{k,i}$ while guaranteeing SLA constraints. The $\tilde{\cdot}^t$ notation highlights that the arrival rate prediction does not refer to the overall arrival rate from the long-term problem, but it is a prediction of the local workload for time slot t. That said, the CA problem can be formulated as:

$$(P_{st}) \qquad \min_{r^t_{k,i}, d^t_{k,i}} \sum_{k \in \mathcal{K}} \left(\rho_i \sum_{t=1}^{n_w} r^t_{k,i} + \delta_i \sum_{t=1}^{n_w} d^t_{k,i} \right)$$

Subject to the conditions:

$$\sum_{\tau=1}^{t} \left(r^\tau_{k,i} + d^\tau_{k,i} \right) + \overline{r}^t_{k,i} + \overline{d}^t_{k,i} > \frac{\tilde{x}^t_{k,i}}{C_i \mu_k} \qquad \forall t \in \mathcal{T}_w, \forall k \in \mathcal{K}, \quad (10)$$

$$\sum_{\tau=1}^{t} (r^\tau_{k,i} + d^\tau_{k,i}) + \overline{r}^t_{k,i} + \overline{d}^t_{k,i} + \geq \tilde{x}^t_{k,i} A_{k,i} \qquad \forall t \in \mathcal{T}_w, \forall k \in \mathcal{K}, \quad (11)$$

$$\sum_{k \in \mathcal{K}} (r^t_{k,i} + \overline{r}^t_{k,i}) \leq W_i \qquad \forall t \in \mathcal{T}_w, \quad (12)$$

$$r^t_{k,i} \geq 0,\ r^t_{k,i} \in \mathbb{N} \qquad \forall t \in \mathcal{T}_w, \forall k \in \mathcal{K}, \quad (13)$$

$$d^t_{k,i} \geq 0,\ d^t_{k,i} \in \mathbb{N} \qquad \forall t \in \mathcal{T}_w, \forall k \in \mathcal{K}. \quad (14)$$

where $A_{k,i} = \dfrac{\overline{R}_{k,i} - D_{k,i}}{(\overline{R}_{k,i} - D_{k,i})\, C_i \mu_k - 1} \geq 0.$

It is worth noticing that problem (P_{st}) shares several similarities with the (P_{lt}); in particular, the objective function is designed to catch the effects of *on-demand* and *reserved* instances on the overall leasing costs, whereas constraints (10), (11) and (12) are semantically equivalent to (3), (4) and (7), respectively. The main differences compared to the (P_{lt}) lie in explicitly considering time (and therefore system state) and not implementing a load sharing policy (i.e., $\tilde{x}^t_{k,i}$, $\forall k, i, t$ are parameters of the model).

More in detail, the inequality set (10) derives from the performance models of Figure 2 and it corresponds to the M/G/1 equilibrium condition; the average response time is given by:

$$R^t_{k,i} = \frac{1}{C_i \mu_k - \frac{\tilde{x}^t_{k,i}}{\overline{r}^t_{k,i} + \overline{d}^t_{k,i} + \sum_{\tau=1}^{t} (r^\tau_{k,i} + d^\tau_{k,i})}} + D_{k,i} \quad (15)$$

Constraints (11), instead, can be obtained after some algebra form the QoS conditions $R^t_{k,i} \leq \overline{R}_k$ and (10) and (15) as follows:

$$\begin{aligned} &\overline{r}^t_{k,i} + \overline{d}^t_{k,i} + \sum_{\tau=1}^{t} \left(r^\tau_{k,i} + d^\tau_{k,i} \right) \leq \\ &(\overline{R}_{k,i} - D_{k,i}) \left[C_i \mu_k \left(\overline{r}^t_{k,i} + \overline{d}^t_{k,i} + \sum_{\tau=1}^{t} \left(r^\tau_{k,i} + d^\tau_{k,i} \right) \right) - \tilde{x}^t_{k,i} \right] \Leftrightarrow \\ &\sum_{\tau=1}^{t} (r^\tau_{k,i} + d^\tau_{k,i}) + \overline{r}^t_{k,i} + \overline{d}^t_{k,i} + \frac{\tilde{x}^t_{k,i} (\overline{R}_{k,i} - D_{k,i})}{1 - (\overline{R}_{k,i} - D_{k,i})\, C_i \mu_k} \geq 0 \end{aligned} \quad (16)$$

Note that, for the reasons discussed in the previous section, it can be safely assumed that $1 - (\overline{R}_{k,i} - D_{k,i})\, C_i \mu_k < 0$.

Lastly, inequalities (12) impose for each time slot t a budget constraint on the overall number of available reserved VMs, that can not be greater than W_i (i.e., the number of VMs for which the SaaS subscribed a long-term contract on provider i).

To conclude notice that, overall, problem (P_{st}) is also a MILP, and it can be efficiently solved by commercial solvers even for large instances (the scalability analysis is discussed in Section 5.5).

4 Solution Algorithm

In this section are presented the two algorithms used to solve the long- and short-term problems, respectively.

Algorithm 1 Request distribution algorithm

1: **procedure** REQUEST DISTRIBUTION
2: **for all** $k \in \mathcal{K}$ **do**
3: $\tilde{\Lambda}_k \leftarrow$ ***GetNextHourPrediction*** (k)
4: **end for**
5: *Solve* (P_{lt})
6: Redirect global workload according to $x_{k,i}$ results
7: **end procedure**

4.1 Solution of the long-term problem

The long term problem algorithm is rather straightforward as it lies directly on the solution of the model presented in Section 3.1. As matter of fact, since the model (P_{lt}) does not consider the state of the system at any point, that is the solutions of two consecutive hours are totally unrelated, the algorithm (as defined in Algorithm 1) does not store information or keep track of previous solutions, ultimately resulting in a lean and easy to implement algorithm.

From the architectural and deployment perspective, within the MODAClouds vision Algorithm 1 is executed by a component hosted by any of the Clouds that make up the execution context. This is necessary because the model around which the algorithm pivots is centralized. The algorithm consists of three main steps: workload prediction, model solution, load balancer manager setting up. It basically invokes a prediction function to forecast the incoming flow of requests for the next hour $(\tilde{\Lambda}_k)$. The flows of real requests, needed for the prediction, are monitored locally to each data center on an hourly basis and then shared with the data center running Algorithm 1. The output of the forecasting function is a vector of future requests that is fed into a solver that computes an optimal solution for the P_{lt} optimization objective defined previously. The result is the partition of the incoming workload across the multiple Cloud providers. The enforcement of the problem solution is performed by properly changing the weights (according to $\frac{x_{k,i}}{\sum_{i' \in \mathcal{I}} x_{k,i'}}$) of the of the MODAClouds multi-Cloud load balancer [37] or of DNS servers of each Cloud provider [38].

4.2 Solution of the short-term problem

The short-term problem is addressed using a controller implementing a receding horizon policy, outlined in Figure 4. A replica of this controller component resides in every Cloud provider (data center) and operates independently from the other providers, unlike the solution for the long-term problem, which is centralized. At each time slot (marked by a clock spike) the monitoring platform on Cloud provider i provides the new workload predictions $(\tilde{x}^1_{k,i}, \ldots, \tilde{x}^{n_w}_{k,i})$ for the current time window $\mathcal{T}_w$ and new estimates for the performance parameters $D_{k,i}$, μ_k. The optimizer component feeds the optimization model using the current application state expressed in terms of allocated VMs. Afterwards, the optimizer harnesses the model to calculate the most suitable number of VM instances to allocate during the whole time window in order to guarantee the arranged SLAs. Finally, the optimizer operates on the running Cloud applications, through IaaS APIs, enacting only the first time slot of the attained allocation plan. Notice that performance parameters are continuously updated at run time in order to capture transient behavior, VMs network and I/O interference [33], and performance variability of the Cloud provider [34].

Algorithm 2 is a high-level description of the receding horizon approach implemented to solve the short-term problem. The algorithm consists in four main steps, iteratively applied to cover the overall time horizon. The first step (lines 2-8) initializes the main model parameters representing the system state and the predicted workload for the considered time window. In particular, the system state is defined by the number of *reserved* and *on-demand* VMs already available for each time slot of the observation window. It is worth noticing that, in order to manage the state of the system $(\overline{r}^t_{k,i}, \overline{d}^t_{k,i})$ in the execution of the algorithm for different time slots, a set of global parameters (i.e., $N^t_{res,k,i}$, $N^t_{ond,k,i}$) has been introduced, which store the number of VM instances, inherited from previous time slots, whose charging period has not yet ended and, therefore, still available at time slot t. The second step (line 9) is the solution of problem P_{st} to optimality using a third-party solver. The third step (line 11) implements the receding horizon paradigm by modifying the application deployment using only the values calculated for the first time slot of the considered time window (to avoid greedy decisions and to exploit a better prediction for the steps not in the near future). Note that most of other literature approaches [6], [12], [30], [39] consider only a single step time instant in the future that can be obtained with $n_w = 1$. Finally, (lines 12-15), the algorithm updates accordingly the system state, $N^{j+t}_{res,k,i}, N^{j+t}_{ond,k,i}$. Since the VMs allocated at time t are available until the end of their charging period, the algorithm only updates the state from t to $t + n_c$. As a consequence, at time slot $t + n_c + 1$ these instances will be switched off, if no longer needed.

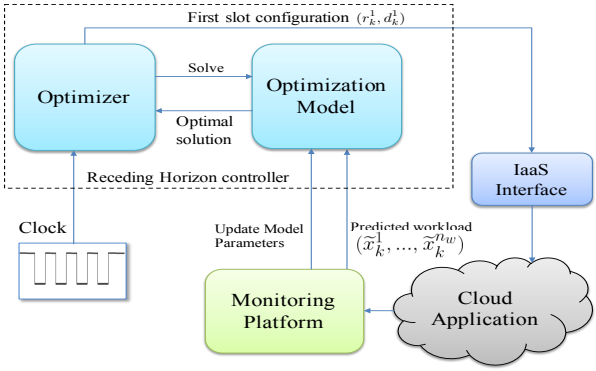

Fig. 4. Receding horizon controller.

5 EXPERIMENTAL CAMPAIGN

In this section, the paper contribution will be compared with the state-of-the-art techniques currently used in Cloud systems, with a focus on the auto-scaling policies that are typically implemented by IaaS cloud providers. In particular, the cost of the different solutions and their ability to meet response time constraints will be investigated. Moreover, the scalability of the whole approach will be also analysed.

Throughout this section, the experimental setup is outlined in Section 5.1 while Section 5.2 provides an insight on the heuristics used for the comparison. In Section 5.3 analytical models are used to discuss costs and the ability to satisfy SLAs of the multiple alternative considered while Section 5.4 evaluates the performance stability exploiting a simulator designed to capture a more realistic scenario where performance may change due to data center resource contention. Finally, Section 5.5 reports the scalability analyses while Section 5.6 concludes the evaluation of this paper research contribution by considering a real system.

Algorithm 2 Receding Horizon Algorithm

```
1: procedure SOLUTION ALGORITHM
2:     for all k ∈ K do                                   } Initialization
3:         for w ← 1, n_w do
4:             x̃^w_{k,i} ← GetPrediction (w, k)
5:             r̄^w_{k,i} ← N^{t+w}_{res,k,i}
6:             d̄^w_{k,i} ← N^{t+w}_{ond,k,i}
7:         end for
8:     end for
9:     Solve (P_st)                                       ← Solving the current model
10:    for all k ∈ K do
11:        Scale (k, r^1_{k,i}, d^1_{k,i})                ← Applying the changes according to the first time slot decisions
12:        for j ← 1, n_c do                              } State update
13:            N^{t+j}_{res,k,i} ← N^{t+j}_{res,k,i} + r^1_{k,i}
14:            N^{t+j}_{ond,k,i} ← N^{t+j}_{ond,k,i} + d^1_{k,i}
15:        end for
16:    end for
17: end procedure
```

5.1 Experimental setup

To assess the performance of alternative resource allocation technuques, scenarios as close as possible to real Cloud environments will be analyzed. A large set of randomly generated instances will be considered. The characteristics of such instances are based on the parameters available in literature, such as [12], [13], [39], or from tests on real applications as in [40], [41]. Table 3 provides an overview of the parameters with their values.

The number of reserved instances will be limited to 10 to introduce the possibility of saturating the available reserved resources and, as a consequence, to consider also the worst case solutions where the on-demand resources are used. It is worth to recall that if reserved instances are not saturated and hence the constraints (7) and (12) can be relaxed, both the long term (P_{lt}) and short term problems (P_{st}) becomes simpler to handle and can be separated in a set of small problems, one for each WS application. With respect to the cost parameters, prices are based on the prices currently charged by IaaS Cloud Providers [2] and are reported in Table 4. In order to model a scenario where the infrastructure is worldwide, instance prices are uniformly random generated within the reported ranges.

TABLE 3
Performance parameters

$D_{k,i}$	μ_k	W
$[0.001, 0.05]$ s	$[200, 400]$ req/s	10 VMs

TABLE 4
Cost parameters

On-Demand	*Reserved*
$[0.060, 1.520]$ \$/h	$[0.024, 0.608]$ \$/h

For the workload Λ_k, instances are generated by using measurements from a large popular Web site (additional details on the site cannot be provided for non-disclosure agreements). The workload exhibits a bi-modal distribution with a peak around 10AM and another in the early afternoon while the night is characterized by low traffic. A different workload for each WS application k is created by scaling the peak value of each request class, as in [11], [40], [42], [43]. Furthermore, random noise is added as in [11]. This extends the original trace and provides the basis for analyzing the behavior of the WS applications for different workload conditions. For the sake of reproducibility, all the script used for generating, adjusting and scaling the workloads are available online as open source [2].

The workload prediction $\widetilde{\Lambda}_k$ is obtained by adding white noise to each sample Λ_k, as in [40], [44], with the noise proportional to the workload intensity Λ_k. For the short time scale problem, the predictions $\tilde{x}^t_{k,i}$ in a real context become less accurate as the number of time slots in the future is increased. Hence, the amount of noise grows with the time step $t \in [1 \dots n_w]$. The choice of applying white noise is consistent with [43] and provides the benefit of being independent from the choice of the prediction technique.

5.2 Heuristics

Before performing the cost-benefit evaluation of this paper contribution, the main heuristics proposed in literature and used to perform a quantitative comparison will be introduced.

Heuristic 1 derives from [12], [13] and is currently implemented by some IaaS providers (see, e.g., Amazon AWS Elastic Beanstalk [14]). The heuristic implements an algorithm for autoscaling the number of instances in order to handle the workload variations. Capacity allocation is performed over the overall time horizon considering an observation window $\mathcal{T}_w$[3] and employs the receding horizon approach by executing only the decisions made for the first time step in $\mathcal{T}_w$. Heuristic 1 fixes the number of instances to allocate in each time slot according to some upper and lower utilization thresholds: In a nutshell, let $\overline{U}_1$ and $\overline{U}_2$ be the lower and upper thresholds respectively. If at time slot t the utilization exceeds $\overline{U}_2$ the number of running VMs at time $t+1$ is suitably increased; otherwise if the considered metric drops under $\overline{U}_1$, a number of VMs, among the oldest ones, is switched off. In this way, the heuristic tries to keep the utilization U^t within the interval $[\overline{U}_1, \overline{U}_2]$.

$$\overline{U}_1 \leq U^t_{H1} \leq \overline{U}_2 \qquad \forall t \in [1..n] \tag{17}$$

As in [40], multiple values for $[\overline{U}_1, \overline{U}_2]$ have been considered. These values will be discussed in deeper details later.

Heuristic 2 is based on a more accurate evaluation of the utilization thresholds: instead of considering fixed values for $\overline{U}_1$ and $\overline{U}_2$, these values are derived from the response time threshold $\overline{R}_k$. In particular, by considering the response time formula (15) and the constraints on the average response time (12) the following condition on the VMs utilization can be obtained:

$$\overline{U}_{k,i} = \frac{\tilde{x}^t_{k,i}}{C_i \mu_k \left(\overline{r}^t_k + \overline{d}^t_k + \sum_{\tau=1}^{t} \left(r^\tau_k + d^\tau_k \right) \right)} = 1 - \frac{1}{C_i \mu_k \left(\overline{R}_k - D_k \right)}$$

This means that $\overline{U}_k$ is the VMs utilization corresponding to the average response time $\overline{R}_k$ at provider i. If we denote with α and β, respectively, the coefficient of lower and upper bound thresholds ($\alpha < \beta$) the utilization thresholds can be defined as:

2. https://github.com/deib-polimi/receding-horizon-workload-scripts

3. Current IaaS providers, e.g., [14], implement a pure reactive mechanism where $\mathcal{T}_w$ includes a single step ahead, here windows with the same size of the receding horizon algorithm are considered for a more fair comparison.

$$\overline{U}_{1,k,i} = \alpha \left(1 - \frac{1}{C_i \mu_k (\overline{R}_k - D_k)}\right); \overline{U}_{2,k,i} = \beta \left(1 - \frac{1}{C_i \mu_k (\overline{R}_k - D_k)}\right)$$

In this way, different thresholds for different WS classes can be obtained. In the experiments the adopted thresholds are characterized by different values of α and β, that will be described later.

5.3 Cost-Benefit Analysis

This paper proposal is evaluated with a twofold goal. First, the cost of the proposed approach is compared with the state-of-the-art solutions previously described to demonstrate that the proposed solution outperforms the available heuristics. Second, the achieved QoS is evaluated to understand how it is affected by the number of time-slots considered in the short-term problem. To this aim two main performance indicators are considered: the overall virtual machines costs and the number of SLA violations.

In this cost-benefit analysis only a single IaaS provider is considered (hence only on Algorithm 2 for the capacity allocation is taken into account). The scenario is based on a time span of 24 hours, with each hour divided into time slots of 10 or 5 minutes, depending on the considered time scale. Incoming traffic can belong either to the *normal traffic model* category or to the *spiky traffic model* one. The two models are based on different traces obtained from real logs. Both traces, (presented in [40]) show a typical diurnal pattern with peaks at 10.00 AM and 2.00 PM. The spiky traffic model shows variations between the number of client requests in two consecutive samples that is far higher than the normal traffic model (that is therefore much smoother). Furthermore, for each workload trace two levels of noise in the prediction are considered: low noise, (corresponding to a more accurate prediction), and high noise. The noise amplitude grows with the observation window, to capture the degradation in the prediction accuracy. Specifically, the noise level in the workload samples $\tilde{x}_k^t, t \in [1 \dots n_w]$ is proportional to the workload Λ_k and grows with t as shown in Table 5. A noise level higher than 45% is not considered as for larger values the prediction of the single sample becomes uncertain and comparable with the sample itself, undermining the effectiveness of the overall approach.

	$T_{slot} = 5min$		$T_{slot} = 10min$	
t	Low noise	High noise	Low noise	High noise
1	10%	10%	10%	15%
2	15%	20%	20%	30%
3	20%	30%	30%	45%
4	25%	40%	40%	—
5	30%	—	—	—

TABLE 5
Noise levels adopted.

In the following analysis the following alternatives are compared:

- The proposed short-term algorithm (*S-t Algorithm*): in the experiments T_{slot} ranges between 5 and 10 minutes and the observation window between 1 and 5 steps.
- Oracle-based algorithm (*Oracle*): this unrealistic algorithm is similar to the S-t Algorithm but has no error in the future traffic prediction (hence this algorithm has no SLA violations and is used as a reference as initially proposed in [11]).
- Heuristic 1 (*Heu1*): has the same time horizon, time step, and VM life span (of one hour for each instantiated VM) as in the S-t Algorithm. The number of instances is determined such that the utilization of each running instance is between some given thresholds. In the experiments $(\overline{U}_1, \overline{U}_2) =$ (40%, 50%), (50%, 60%), and (60%, 80%) are considered as in [13], [40].
- Heuristic 2 (*Heu2*): the threshold coefficients considered are $(\alpha, \beta) = (0.9, 1.2)$ and $(0.8, 1.3)$.

The first analysis focuses on the costs of the previously described alternatives. In this study 10 different classes of requests are considered and the mean solution cost is shown (the cost is evaluated for a 24 hours time horizon over multiple test executions). In particular, three tests for each configuration of time scale, workload type, noise and thresholds are carried out, for a total number of 72 runs. For each run the cost is computed considering the whole time horizon. Due to space limits only the spiky workload case is described while a more detailed description of the experimental results can be found in [45].

Figures 5 and 6 refer to a time scale of 5 minutes while Figures 7 and 8 refer to a 10 minutes time scale. We recall that in some of the figures the number of time slots considered is less than 5 because, in the case of high noise level and for a long time scale, the noise level would exceed 45% and would make the prediction highly unreliable. It is worth noticing that the proposed approach achieves performance similar to the Oracle solution for every considered scenario. In particular, for the 5 minutes time scale, the proposed approach is the second best performing solution, outperformed just by the Oracle solution. The impact of the noise is evident if we consider that the distance between the Oracle and the proposed solution grows as the noise level increases.

Heuristic 2 is the best solution for the 10 minutes time scale, since the thresholds computed take into account the Web requests SLA. However, it is worth anticipating that the low costs of Heuristic 2 have a major impact in terms of SLA violations as we will show in the following of the analysis. On the other hand, the Oracle never violates the SLAs (indeed, no unexpected conditions can arise). Finally, Heuristic 1 turns out to be the worst solution in the comparison. However, it gradually improves with the growth of the utilization thresholds.

Beside the cost of the solution, the number of SLA violations during the 24 hours time horizon is also evaluated (the number of SLA violations is the number of times during a time slot where the average response time is above the threshold). The goal of this analysis is to understand if the receding horizon approach reduces the violations with respect to single step optimization (that is when $n_w = 1$), which characterizes most of other literature approaches. Tables 7, 8, 10 and 9 reported in Appendix show the results of this analysis. We clearly see that increasing the observation window size n_w, the percentage of SLA violations for the proposed solution decreases. Furthermore, the proposed technique clearly outperforms Heuristic 2, both (0.9, 1.2) and (0.8, 1.3), even for the spiky and highly noisy workloads, where Heuristic 2 provides lower costs at the price of a higher number of SLA violations (from 13.530% to 17.245% depending on the time scale and noise level). We can then conclude that the receding horizon technique improves the performance of the system in terms of SLA violations. On the other hand, Heuristic 1 is very conservative: it satisfies the QoS constraints almost in any condition of traffic and noise, but it is more expensive than the proposed approach.

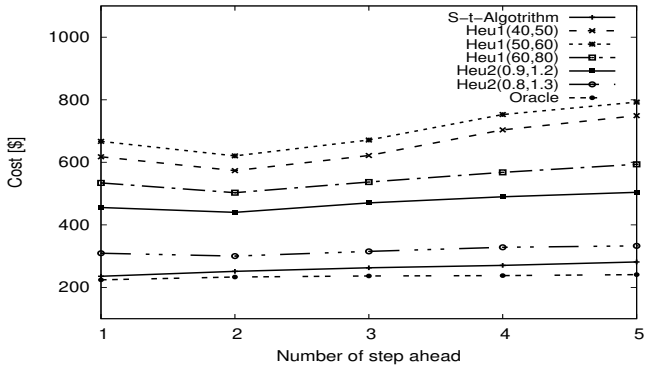

Fig. 5. Solution cost, $T_{slot} = 5min$, spiky traffic and low noise level.

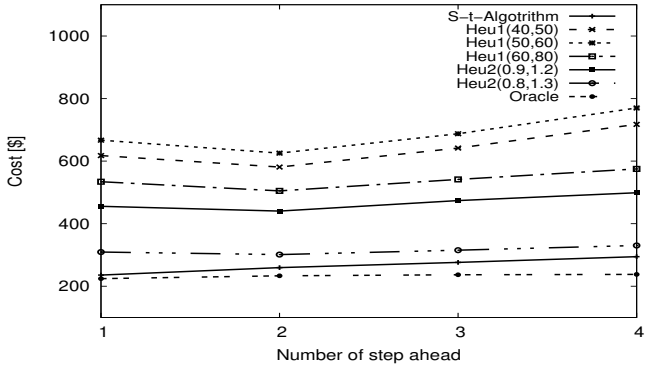

Fig. 6. Solution cost, $T_{slot} = 5min$, spiky traffic and high noise level.

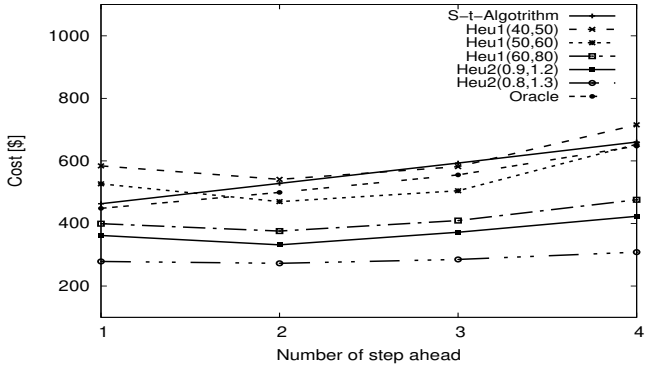

Fig. 7. Solution cost, $T_{slot} = 10min$, spiky traffic and low noise level.

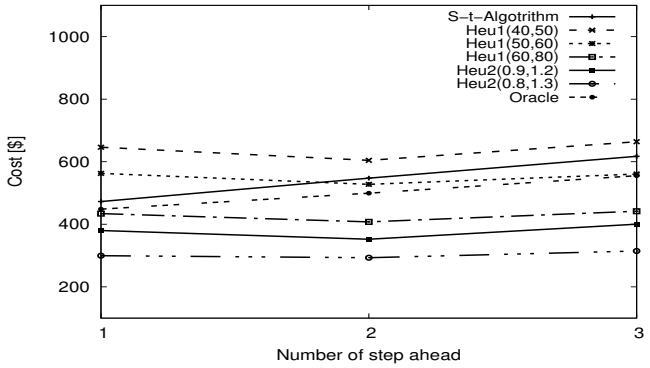

Fig. 8. Solution cost, $T_{slot} = 10min$, spiky traffic and high noise level.

Comparing the results for the two considered time scales, we observe that the 5 minutes time scale provides better performance in terms of SLA violations as violations occurs for 0.556% of the time (8 minutes over 24 hours) for the normal traffic and 1.065% for the spiky one (see Tab. 7 in Appendix). For the 10 minutes interval, the SLA violations are in the range 3.102% – 4.259%, depending on the noise level (which is still far lower, compared to the worst case of the heuristics that we recall reaches 17.245%). Furthermore, if two or more forward steps are used, there is an additional performance gain for example, in the spiky traffic the SLA violations drops from 1.065% – 4.259% to 0.428% – 1.204% (depending on the time scale and noise level) This means that, from the SLA violations point of view, the proposed algorithm increases significantly its performance with the adoption of at least two forward steps. Finally, the analysis demonstrates that the proposed algorithm steadily reduces SLA violations as the the observation windows increases, reaching 0.208% – 0.625% of SLA violations.

5.4 System simulation

Although a first confirmation of the viability of the proposed approach is provided by the analytical evaluation of the proposed algorithm (i.e., based on M/G/1 formula) reported in the previous section, the effect of the interaction among VMs that occurs in a real data center cannot be taken into account by an analytical-only validation. In this section a simulator specifically designed to capture the variable performance of real data centers is used. The aim is to evaluate the proposed approach in a more realistic scenario and to consider multiple providers/data centers. In the following, first the simulator and its setup are described, after which an in-depth discussion of the results obtained through simulation is provided.

5.4.1 Simulation Setup

The proposed simulator implements the data center model (illustrated in Figure 2) described in Section 2. In particular, a reference scenario with three heterogeneous data centers has been considered, with the global processing power distributed over the data centers according to the following percentages: $\{50\%, 25\%, 25\%\}$, so that the first data center has a processing capacity double w.r.t. the others. The incoming workload is partitioned by the long-term algorithm according to the processing capacity of each data center while the short-term algorithm handles the VMs management in each data center.

To validate our proposal a discrete event simulator has been used, which is based on the Omnet++ framework [46] and has been developed ad-hoc for this purpose. In particular, in order to capture the performance degradation that appears randomly in Cloud data centers due to resource contention and imperfect performance isolation between VMs, Random Environments (REs) [15] are included in the simulator. REs are Markov chain-based descriptions of time-varying system operational conditions that evolve independently of the system state. Hence, REs are natural descriptions for exogenous variability in a Cloud environment [15], [47] and have been successfully used also for analyzing the performance of servers adopting dynamic voltage-scaling technologies that change CPU frequency over time to reduce the energy consumption [48].

REs are implemented within the simulator with two stages to model the variability of performance of a virtual hardware due to resource contention (see Figure 9). Under resource contention, individual VMs are characterized by the service rate μ_k^{slow} and delay D_k^{slow}, while under normal operation VMs are characterized by parameters $\mu_k^{fast} > \mu_k^{slow}$ and $D_k^{fast} < D_k^{slow}$. A transition probability between the two stages is considered: p^{fast} is the transition probability from the *slow* to *fast* state while p^{slow} is the probability of the opposite transition.

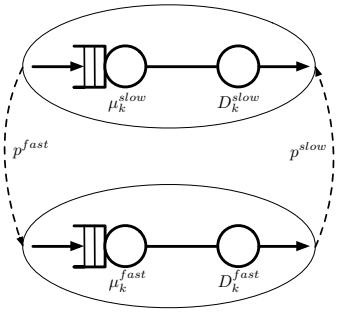

Fig. 9. Random Environment modeling Cloud resource contention.

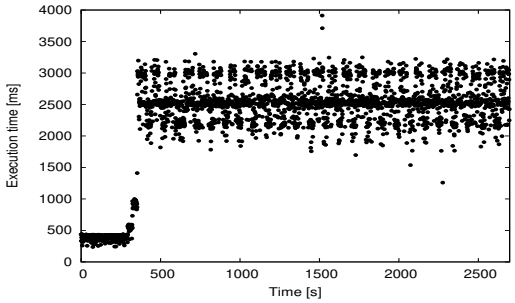

Fig. 10. Execution time under CPU steal effect in Amazon EC2 medium instance

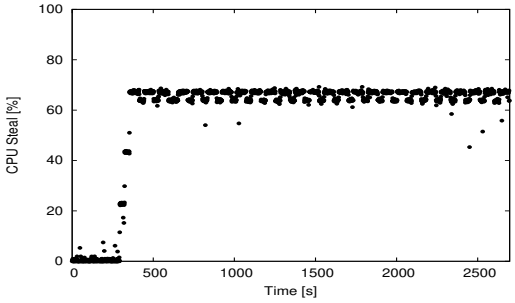

Fig. 11. CPU steal in Amazon EC2 medium instance.

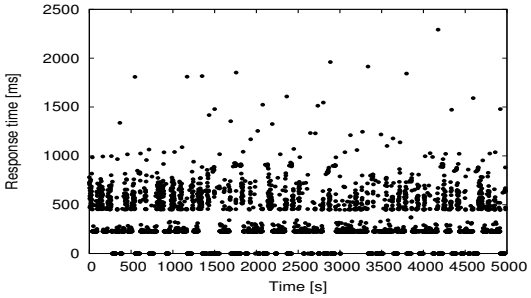

Fig. 12. Performance degradation in a multi VM web server application.

In order to model these effects in the simulator, data from several experiments on applications running on an Amazon EC2 infrastructure were collected. To this aim, a computationally-intensive application has been executed on VMs of different sizes, monitoring the response time of the application as well as the system parameters. In particular, the CPU utilization and CPU steal were considered (this latter is the fraction of VM CPU time that is claimed by the hypervisor). According to the experimental results, when the application is executed on large VM instances there is no performance degradation and no CPU stealing from the hypervisor. On the other hand, when medium or smaller VM instances are used, the hypervisor introduces a limitation in the amount of computational resources that can be used by a single VM. Specifically, as evident from Figures 10 and 11, under constant incoming workload, after a period with full processing power, the CPU consumption of the VM is capped by the hypervisor, the VM performance degrades and the CPU steal parameter increases. For an application that is not completely CPU-bound, such as a web-based application where a load balancer distributes the load across multiple VMs, the effect of resource capping is less evident, with the response time assuming a bimodal distribution, as suggested by the samples in Figure 12. In this case, the limitation on the CPU demands is activated in an intermittent way, determining an alternating succession of normal and degraded performance that is easily modeled by the REs model of our simulator.

The parameter of the REs in the simulator are obtained through experiments on the real system. Table 6 reports the resulting values. In particular, the impact of congestion on the VM performance (both in terms of processing time and of delay) is quantified using the performance degradation ratio while the transitions between $slow$ and $fast$ states are modeled as exponentially distributed time intervals characterized by their average value. The parameters for the data center in a state with no resource contention (D^{fast} and μ^{fast}) are the same used in the previous experiments and can be found in Table 3.

TABLE 6
Simulator parameters

parameter	value
RE parameters	
$\frac{\mu^{fast}}{\mu^{slow}} = \frac{D^{slow}}{D^{fast}}$	2.42
$\overline{T}(fast \rightarrow slow)$	994.05 [s]
$\overline{T}(slow \rightarrow fast)$	694.80 [s]
Data center parameters	
Q	5-40
To	$5, 10 \times \frac{1}{\mu^{fast}}$

An additional element that is captured by the simulator is the process of client requests dropped from the waiting queue. We consider that client requests may leave the system without being served for the following two reasons: (1) because of the expiration of a timeout To as the user is not willing to wait for the completion of the request and (2) because the WS buffer queue size is a finite value Q and only a limited number of requests may be waiting for service. This latter functionality is typically configured in web servers such as Apache httpd [49] to avoid thrashing conditions. In the performed experiments, several scenarios have been explored for different combinations of To and Q, as detailed in the following subsection. A summary of the parameters used for simulations is provided in Table 6.

5.4.2 Sensitivity to queue length and timeout

The simulator introduced in the previous section has been used to evaluate the number of dropped requests as well as the number of SLA violations for different values of the window $\mathcal{T}_w$ and for different values of queue length Q and timeout To. Specifically, two scenarios for the timeout parameter has been considered, that is, a *short timeout* scenario where the timeout is five times the response time without resource contention $\frac{1}{\mu^{fast}} + D^{fast}$ and a *long timeout* where the timeout is ten times that value. For each scenario multiple queue lengths Q and observation window sizes $\mathcal{T}_w$ have been considered.

The results of these experiments for the two considered scenarios are reported in Appendix by Table 7. With respect to the short timeout scenario, three interesting results can be observed. First, for every considered combination of parameters, no SLA violation occur. Focusing on the percentage of dropped requests, it can be observed that the values range from 0.3% to 1.8%.

The second finding is that the percentage of dropped requests decreases as the queue length grows. This result is easily explained by considering that with longer queues the system is able to better manage incoming peaks of requests, without the need to reject some of them. Finally, the number of dropped requests decreases also with increasing size of the observation window used by the receding horizon algorithm. This result confirms that the proposed solution can handle effectively the Cloud workloads with an auto-scaling mechanism that is more effective as the observation window increases.

What has been found experimenting with a short timeout is basically confirmed by the experiments with the long timeout scenario. However, comparing the results of the two scenarios, it can be observed that the percentage of dropped requests for the long timeout case ranges from 4.2% to 0.7%, which is significantly higher than the results of the short timeout alternative. This result can be explained considering that, in the case of a long timeout, requests stay within the system for longer periods and, due to the processor-sharing nature of the model, consume resources that may delay other jobs. On the other hand, in the case of a short timeout, long requests tend to be dropped from the system after a short time, thus having a smaller opportunity to consume resources.

Furthermore, it is worth noting that, when the queue length grows beyond 30 requests, besides the reduction in the number of dropped requests, a small amount of SLA violations can be experienced (below 0.1%). This result can be explained by considering that the longer the queue length, the higher the number of requests within the system, resulting in a growth in the response time, which eventually determines the SLA violations.

5.5 Scalability Analysis

This section presents the results of a scalability analysis that has been performed for both the long and the short-term problems. In particular, in order to consider problems of increasing sizes, the number of classes has been varied from 40 up to 160 with step 20. It is worth noting that in the scalability analysis of the short-term problem also the number of steps ahead becomes a degree of freedom. In particular, it has been varied between 1 and 5. The experiments has been conducted on a server based on an Intel Xeon Silver 4114 2.20 GHz processor and 48 GB of ram and using version 10.1.0 of the CPLEX solver. Figure 13 shows, for problem instances of different sizes, the computational times

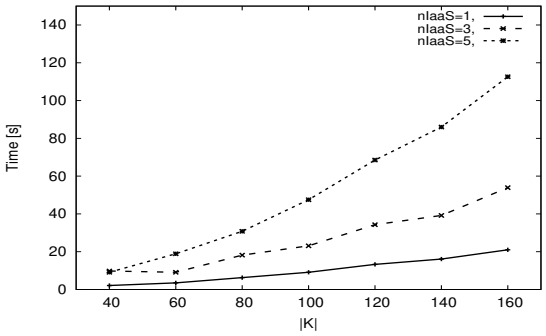

Fig. 13. Long-term Problem scalability.

in seconds (averaged over ten different runs) required to solve the long-term capacity allocation problem, which is, in all the cases, within very few minutes. Therefore the problem can be solved on an hourly basis, i.e. the granularity considered for the long-term problem. Figure 13 also shows that the proposed method scales linearly with respect to the number of classes. Indeed, the coefficient of determination with respect to a linear function is $\mathcal{R}^2 = 0.978$ when $|\mathcal{I}| = 1$, $\mathcal{R}^2 = 0.953$ when $|\mathcal{I}| = 3$ and $\mathcal{R}^2 = 0.978$ when $|\mathcal{I}| = 5$.

Figure 14 plots the execution time, for different problem sizes and averaged over ten runs, of the short-term problem, varying the number of steps ahead within the range $[1, 5]$. The Figure shows that the proposed method scales linearly with respect to the number of classes. The values of the determination coefficient with respect to a linear function is $\mathcal{R}^2 = 0.977$ in case of $n_w = 1$, $\mathcal{R}^2 = 0.978$ in case of $n_w = 3$ and $\mathcal{R}^2 = 0.981$ in case of $n_w = 5$. Moreover, this analysis demonstrates that the short-term problem can be solved according to a time scale of 5/10 minutes, even in the case in which $|\mathcal{K}| = 160$ and $|\mathcal{I}| = 5$, which can be considered, in practice, a very complex case. Indeed, the short term problem solution can be computed in less then two minutes and there is time (about three minutes) to provision, possibly, the additional VMs identified for the next control time horizon. However, it is quite unlikely in practice for a Cloud customer to have more than few tens of different application classes. Note that, considering additional time slots does not change significantly the optimization time.

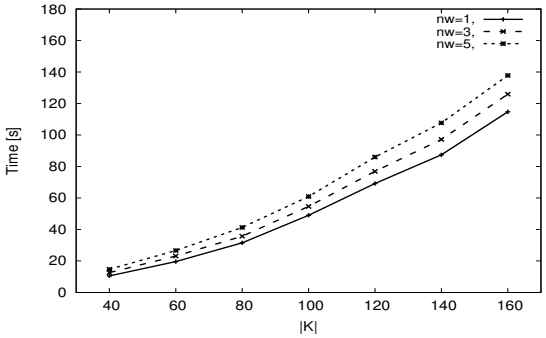

Fig. 14. Short-term Problem scalability.

5.6 Prototype Environment Analysis

The experimental analysis presented in this section has been performed considering the *Modelio Constellation* modeling platform

developed by Softeam within the MODAClouds project [50]. Modelio Constellation is a web-based software-as-a-service modeling environment, which includes four main components. The *Administration Server* provides users with a GUI through which the available projects and their configuration can be retrieved, modified and updated. The *Administration Database* is used to store the access permission policies. The *SVNAgent* uses SVN to provide versioning, sharing and conflict management with the aim of enabling multiple users to work simultaneously on the same project. To offload the previous component from some of the burden, the *HTTPAgent* component provides read-only access to the models. Constellation is subject to a variable workload during a day.

In order to reduce the complexity of the Constellation case study and given the fact that the 80% of the requests have been found to be SVN reads, a simplified scenario has been considered for the evaluation of the short term algorithm, in which the only *HTTPAgent* was deployed in a dedicated VM. Along this path, the resource demand of the *HTTPAgent* was measured and found to be around 40 ms (in particular $D_k = 5$ ms, $\frac{1}{\mu_k} = 35$ ms).

In order to evaluate the system, two different workloads have been considered: (1) a ramp-like workload (Figure 15) with a peak around 30 requests/s was and (2) a workload from real users (Figure 16) spanning 24 hours and obtained from the logs of a pre-production environment, which was basically a bi-modal workload with 100 requests/s at its peaks, located in the central part of the day. In this second scenario, the original 24 hour-lasting workload was shrunk to 4 hours in order to reduce the duration of each experiment, making sure to scale down also the workload peak, to keep the original workload variations.

The validation experiments tested the capability of the short term algorithm to react to workload fluctuations considering the SOFTEAM *HTTPAgent* component deployed on Amazon m3.large VMs. Apache JMeter was as workload injector. An average response time QoS constraint equal to 560 ms was set and our receding horizon algorithm used a 5 steps ahead control with a time period of 5 minutes. Workload predictions were obtained by using an ARIMA model while service demand estimates were obtained through the Extended Regression for Processor Sharing resources (ERPS) method [51], acting at 10s time scale.

The short term algorithm started up to 11 VM instances in the case of the ramp-like workload and up to 10 VM instances at the second peak of the bimodal workload. Figures 17 and 18 show the requests response time for the two considered workloads, along with the response time threshold that was set in order to highlight violations. In the very worst case (ramp-like workload), the percentage of violations of the average response time measured at runtime was 1.9% when the average is computed over time windows of 10 seconds, while it was 0% when the average is computed over the control time window of 5 minutes. In the ramp workload scenario the number of allocated VMs is step wise, while in the bimodal workload case the number of allocated VMs follows the workload trend.

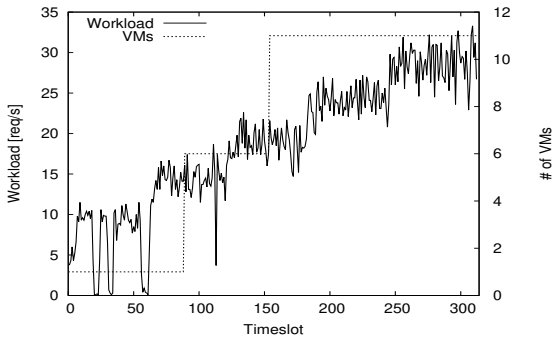

Fig. 15. VMs allocation, 5 minutes time scale, ramp workload and low noise level.

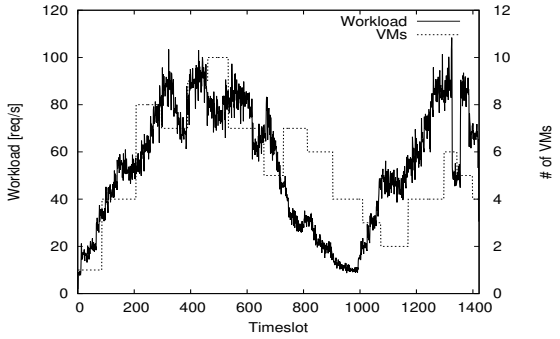

Fig. 16. VMs allocation, 5 minutes time scale, real bimodal workload and low noise level.

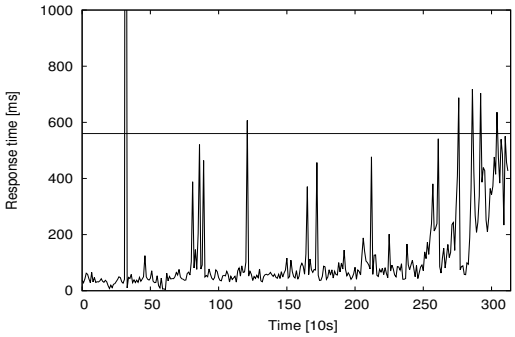

Fig. 17. Response times, 5 minutes time scale, ramp workload and low noise level.

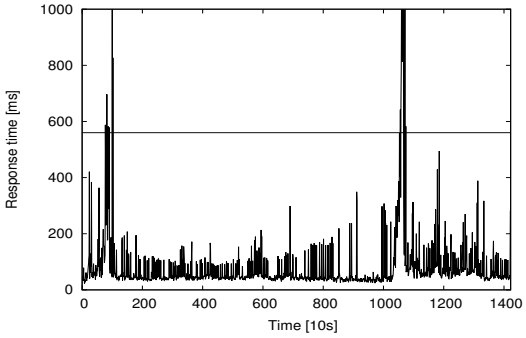

Fig. 18. Response times, 5 minutes time scale, real bimodal workload and low noise level.

6 Related Work

As Cloud computing is a promising technology, rapidly growing and appealing for industries, there is a large corpus of literature devoted to this topic. A large number of studies consider resource management in a single data center [6], [7], [43], [52], [53], [54], [55] while other studies focus on a more complex scenario where multiple data centers (and possibly providers) are involved [5], [56], [57], [58], [59]. For example, [5] focuses on placing cloud services over a distributed infrastructure by introducing affinity (from a QoS point of view) between services and geographic regions, data centers, up to single hosts. This paper contribution fits in the category of multi data center scenarios, but, unlike [5] presents a more complex model of the data centers that include performance (and SLA compliance) with economic concerns. Furthermore, this paper proposal operates on two different time scales combining workload allocation and resource management

from a double time-scale perspective.

Another aspect that differentiate the existing literature is the actor performing the resource allocation that may range from a IaaS cloud provider managing the allocation of VMs [60], [61] to a PaaS/SaaS where the workload is allocated over the infrastructure [53], [62], possibly considering also the scaling up through the creation of new instances. This research follows this latter approach applying it to both the switching ON/OFF of VMs in a single data center (that is the main scope of [53]) and to the distribution of workload over multiple data centers.

The techniques used to approach the resource allocation problem propose a plethora of heuristics ranging from Ant Colony Optimization [6] to Genetic Algorithms [63] or rely on hierarchical approaches [43], [57], [64] for scalability reasons. A different approach is to focus more on a mathematical formulation of the problem [52], [53], [60], [65], [66], aiming to a closed-form solution or relying on external solvers. In the proposed approach a mathematical formulation of the problem (with a multi-class queuing system inspired by [66]) is merged with the proposal of an algorithm that takes into account also the scalability issues in the solution of the underlying optimization problem. It is worth to remark that a qualifying point of the present proposal is to split the overall management problem in two sub-problems at different time scales.

Several aspects of the service requests are taken into account in the present research. For example, as in [52], the presence of timeouts is considered but, rather than proposing a mechanism for adapting the timeout to avoid overload, the timeout is considered as a parameter of the experimental scenario. The effect of timeout values is analyzed to demonstrate the stability of the proposed solution for different setups. Another significant characteristic of the proposed model is considering the time divided into time-slots (as in [53]). In [53], dividing time into discrete intervals supports a dynamic programming approach where an input of the optimization problem is the system status in the previous time step and is a suitable model to capture the nature of the VM allocation contracts (where a VM is purchased on an hourly basis). However, the present paper extends the basic approach of [53] through a time-receding algorithm that considers multiple future time steps at once and provides a clear benefit in terms of achieved QoS. The receding horizon approach used for the short-term problem was first proposed in [10]; however, in the present paper the receding-horizon technique is integrated with a long-term workload management strategy and the model is extended by embracing a multi-cloud vision.

Beside the effort to consider QoS in Cloud systems (considering both performance and SLA violations), common to most literature, the present study takes directly into account also the economic aspects of the problem of Cloud management, as in [7], [67], [68], [69]. However, several existing papers rely on a simplified economic model for example directly related to energy consumption [7] or based on the knowledge of a standard per-request cost [68]. The economic model in this paper is more complex as it considers the dynamic aspects of VMs allocation, that are to be purchased for periods of time several orders of magnitude higher compared to the service time of a single request.

An additional qualifying point of the present study is that the impact of resource contention on the infrastructure and its effect on performance and SLA violations are consiered, similarly to [5], [6], [7], [8]. In particular, the proposed model to handle VM interactions and workload burstiness is inspired by [8], but is extended from a simple performance model in a single data center, to a core part of a simulator that can evaluate the QoS at the level of multiple data centers, demonstrating the robustness of the proposed algorithms. The model includes also a prediction step as in [4]. However, the present study goes beyond the simple application of standard prediction algorithms ([4] exploits Markov chains model) providing a detailed analysis of the impact of the prediction error (e.g., due to noisy samples) on the overall performance of the data center management.

Finally, he present research is characterized by an holistic approach to the analysis that is almost non-existent in literature. In particular, state-of-the-art solvers as in [53], [67] to evaluate the problem from a mathematical point of view, are combined with the implementation of a full-featured data center simulator as in [62], [70] to capture elements of the model such as resource contention that are not easy to include in the mathematical model (such as the effect of resource contention); furthermore experiments on a real infrastructure as in [71] are carried out to provide an additional validation of our proposal. This extensive evaluation demonstrates that the proposed approach can cope with a wide range of scenarios ensuring better performance compared to existing solutions both in terms of QoS (by satisfying the SLAs) and in terms of economic cost.

7 Conclusions

In this work a novel dual time scale receding horizon resource allocation technique has been proposed. The approach is able to minimize the execution cost of Cloud applications guaranteeing the respect of multi-class SLAs. An extensive analysis that takes into account multiple factors as different workloads and system configurations has been provided demonstrating that this paper solution outperforms major techniques available in the literature or currently used by IaaS providers.

When compared with an Oracle with perfect knowledge of the future the cost gap is about 7% on average. With respect to heuristic solutions, cost savings range in [30, 80]%. If the multiple solutions are compared in terms of SLA violations the benefit of the adoption of the receding horizon control is evident, since in the very worst case response time violation occurs in 4.259% of times (versus 17.245% of other approaches). Furthermore SLA violations can be reduced to no more 0.625% exploiting the receding horizon characteristics, while the alternatives have worst cases of up to 16.042% of SLA violations even with longer time windows.

Finally, results have shown that in normal traffic conditions, the best time scale length is 10 minutes while with spiky workloads the receding horizon is more effective considering a more fine grained time scale, i.e., 5 minutes.

Future work will be devoted to the development of an adaptive approach that will be able to switch between different time scales according to the workload conditions.

Acknowledgement

The research reported in this article is partially supported by the European Commission grant no. FP7-ICT-2011-8-318484 (MODAClouds).

Danilo Ardagna is an Associate Professor at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano, Milan, Italy. He received the Ph.D. degree in Computer

Engineering from Politecnico di Milano in 2004. His work focuses on performance modelling of software systems and on the design, prototype and evaluation of optimization algorithms for resource management and planning of Cloud and big data systems.

Michele Ciavotta received the Ph.D. degree in automation and computer science from Roma Tre, Italy in 2008. He is researcher at the University of Milano-Bicocca since 2017. His research work focus on modeling and optimization of highly constrained combinatorial problems mainly arising in the fields scheduling and resource management of distributed systems.

Riccardo Lancellotti received the Ph.D. in computer engineering from the University of Roma "Tor Vergata" in 2003. He is a researcher at the University of Modena and Reggio Emilia since 2005. His research interests include geographically distributed systems, Cloud computing and social networks. For additional information: http://web.ing.unimo.it/rlancellotti/

Michele Guerriero is a Ph.D. candidate at Politecnico di Milano, Italy. His main research interests concern software engineering methods and advanced software architectures in the context of Cloud computing and Big Data. In particular, his research addresses model-driven engineering methods for modern data-intensive applications, with a special interest on the deployment and data privacy issues.

REFERENCES

[1] Google Inc., "Google inc." [Online]. Available: www.google.com/about/company/

[2] Amazon Inc., "Amazon Web Services," http://aws.amazon.com/.

[3] Microsoft, "Microsoft corporation," http://www.microsoft.com/.

[4] L. Wei, C. H. Foh, B. He, and J. Cai, "Towards Efficient Resource Allocation for Heterogeneous Workloads in IaaS Clouds," *IEEE Trans. on Cloud Computing*, vol. 6, no. 1, pp. 264–275, 2018.

[5] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and Placement of Structured Cloud Services," *IEEE Trans. on Cloud Computing*, vol. 4, no. Iii, pp. 1–10, 2016.

[6] H. Zhao, J. Wang, F. Liu, Q. Wang, W. Zhang, and Q. Zheng, "Power-aware and Performance-guaranteed Virtual Machine Placement in the Cloud," *IEEE Trans. on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1385–1400, 2018.

[7] X. Jin, F. Zhang, L. Wang, S. Hu, B. Zhou, and Z. Liu, "Joint Optimization of Operational Cost and Performance Interference in Cloud Data Centers," *IEEE Trans. on Cloud Computing*, vol. 5, no. 4, pp. 697–711, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7134716/

[8] G. Casale, N. Mi, L. Cherkasova, and E. Smirni, "Dealing with burstiness in multi-tier applications: Models and their parameterization," *IEEE Trans. on Software Engineering*, vol. 38, no. 5, pp. 1040 –1053, sept.-oct. 2012.

[9] Amazon Inc., "What Is Amazon EC2 Auto Scaling?" https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html.

[10] D. Ardagna, M. Ciavotta, and R. Lancellotti, "A receding horizon approach for the runtime management of iaas cloud systems," in *SYNASC 2014 Proceedings*, 2014.

[11] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[12] A. Wolke and G. Meixner, "Twospot: A cloud platform for scaling out web applications dynamically," in *ServiceWave*, 2010.

[13] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D.Gmach, R. Gardner, T. Christian, and L. Cherkasova:, "1000 islands: An integrated approach to resource management for virtualized data centers," *J. of Cluster Computing*, vol. 12, no. 1, pp. 45–57, 2009.

[14] Amazon Inc., "AWS Elastic Beanstalk," http://aws.amazon.com/elasticbeanstalk/.

[15] G. Casale and M. Tribastone, "Modelling exogenous variability in cloud deployments," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 73–82, Apr. 2013.

[16] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, and C. Sheridan, "Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds," in *MISE 2012 Proceedings*, 2012.

[17] "MODAClouds: MOdel-Driven Approach for design and execution of applications on multiple Clouds." http://www.modaclouds.eu.

[18] Amazon Inc., "Amazon Elastic Cloud," http://aws.amazon.com/ec2/.

[19] ——, "Elastic Load Balancing," http://aws.amazon.com/elasticloadbalancing/.

[20] M. Stecca, L. Bazzucco, and M. Maresca, "Sticky session support in auto scaling iaas systems," in *2011 IEEE World Congress on Services*, 2011.

[21] S. Casolari and M. Colajanni, "On the selection of models for runtime prediction of system resources," *Autonomic Systems, Springer*, 2010.

[22] D. Ardagna, S. Casolari, M. Colajanni, and B. Panicucci, "Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 6, pp. 796–808, 2012.

[23] S. Casolari and M. Colajanni, "Short-term prediction models for server management in internet-based contexts," *Decis. Support Syst.*, vol. 48, no. 1, pp. 212–223, Dec. 2009.

[24] D. Ardagna, M. Ciavotta, and M. Passacantando, "Generalized nash equilibria for the service provisioning problem in multi-cloud systems," *IEEE Trans. on Services Computing*, vol. 10, no. 3, Sept. 2015.

[25] M. Scavuzzo, E. Di Nitto, and D. Ardagna, "Experiences and challenges in building a data intensive system for data migration," *Empirical Software Engineering*, 2017.

[26] L. Zhang, X. Meng, S. Meng, and J. Tan, "K-scope: Online performance tracking for dynamic cloud applications," in *ICAC 2013 Proceedings*, 2013.

[27] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, "Loosely coupled coordinated management in virtualized data centers," *Cluster Computing*, vol. 14, no. 3, pp. 259–274, 2011.

[28] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi, "Analytic modeling of multitier Internet applications," *ACM Transaction on Web*, vol. 1, no. 1, January 2007.

[29] Z. Liu, M. Squillante, and J. L. Wolf, "On Maximizing Service-Level-Agreement Profits," in *Proc. of ACM Eletronic Commerce Conference*, October 2001.

[30] X. Wang, Z. Du, Y. Chen, and S. Li, "Virtualization-based autonomic resource management for multi-tier Web applications in shared data center," *J. Syst. Softw.*, vol. 81, no. 9, pp. 1591–1608, 2008.

[31] V. Gupta and M. Harchol-Balter, "Self-adaptive Admission Control Policies for Resource-sharing systems," in *SIGMETRICS*, 2009.

[32] T. Nowicki, M. S. Squillante, and C. W. Wu, "Fundamentals of dynamic decentralized optimization in autonomic computing systems," in *Self-star Properties in Complex Information Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 204–218.

[33] Y. Mei, L. Liu, X. Pu, S. Sivathanu, and X. Dong, "Performance analysis of network i/o workloads in virtualized data centers," *IEEE Trans. on Services Computing*, vol. 6, no. 1, pp. 48–63, 2013.

[34] A. Croll, "Cloud performance from the end user perspective," http://www.bitcurrent.com/download/cloud-performance-from-the-end-user-perspective/.

[35] G. Casale, W. Wang, M. Miglierina, and V. I. Munteanu, "D6.3.2 Monitoring platform final release," http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds_D6.3.2_MonitoringPlatformFinalRelease.pdf.

[36] Wook Hyun Kwon, Soo H. Han, *Receding Horizon Predictive Control: Model Predictive Control for State Models*. Springer, 2005.

[37] G. Iuhasz, P. Jamshidi, W. Wang, and G. Casale, "Load balancing for multi-cloud," *Model-Driven Development and Operation of Multi-Cloud Applications The MODAClouds Approach*, p. 53, 2017.

[38] "NGINX," https://www.nginx.com/resources/admin-guide/load-balancer.

[39] J. Almeida, V. Almeida, D. Ardagna, I. Cunha, C. Francalanci, and M. Trubian, "Joint admission control and resource allocation in virtualized servers," *J. of Parallel and Distributed Computing*, vol. 70, no. 4, pp. 344 – 362, 2010.

[40] D. Ardagna, S. Casolari, M. Colajanni, and B. Panicucci, "Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems," *J. of Parallel and Distributed Computing*, vol. 72, no. 6, pp. 796 – 808, 2012.

[41] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang, "Energy-aware autonomic resource allocation in multitier virtualized environments," *IEEE Trans. Services Computing*, vol. 5, no. 1, pp. 2–19, 2012.

[42] D. Ardagna, S. Casolari, and B. Panicucci, "Flexible distributed capacity allocation and load redirect algorithms for cloud systems," in *IEEE CLOUD 2011 Proceedings*, 2011.

[43] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, and L. Zhang, "A hierarchical approach for the resource management of very large cloud platforms," *IEEE Trans. on Dependable and Secure Computing*, vol. 10, no. 5, pp. 253–272, 2013.

[44] D. Kusic, N. Kandasamy, and G. Jiang, "Approximation modeling for the online performance management of distributed computing systems," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 5, pp. 1221 –1233, oct. 2008.

[45] D. Ardagna and M. Ciavotta, "Receding Horizon Auto-Scaling Algorithm for IaaS Cloud Systems. Politecnico di Milano Technical Report 2014.5," http://home.deib.polimi.it/ardagna/Cloud2014.pdf.

[46] "OMNeT++ Discrete Event Simulation System," 2014, - http://www.omnetpp.org.

[47] G. Casale and M. Tribastone, "Fluid analysis of queueing in two-stage random environments," in *QEST*, 2011.

[48] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Perform. Eval.*, pp. 1155–1171, 2010.

[49] "Apache Server," http://httpd.apache.org.

[50] "Modelio Project Management Server Constellation," https://link.springer.com/chapter/10.1007/978-3-319-46031-4_12.

[51] J. F. Perez, G. Casale, and S. Pacheco-Sanchez, "Estimating computational requirements in multi-threaded applications," *IEEE Trans. on Software Engineering*, vol. 41, no. 3, pp. 264–278, March 2015.

[52] S. Homsi, S. Liu, G. A. Chaparro-Baquero, O. Bai, S. Ren, and G. Quan, "Workload consolidation for cloud data centers with guaranteed QoS using request reneging," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2103–2116, 2017.

[53] L. Shi, Y. Shi, X. Wei, X. Ding, and Z. Wei, "Cost Minimization Algorithms for Data Center Management," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 60–71, 2017.

[54] B. Wanis, N. Samaan, and A. Karmouch, "Efficient Modeling and Demand Allocation for Differentiated Cloud Virtual-Network as-a Service Offerings," *IEEE Trans. on Cloud Computing*, vol. 4, no. 4, pp. 376–391, 2016.

[55] A. Paya and D. C. Marinescu, "Energy-Aware Load Balancing and Application Scaling for the Cloud Ecosystem," *IEEE Trans. on Cloud Computing*, vol. 5, no. 1, pp. 15–27, 2017.

[56] M. Mechtri, M. Hadji, and D. Zeghlache, "Exact and Heuristic Resource Mapping Algorithms for Distributed and Hybrid Clouds," *IEEE Trans. on Cloud Computing*, vol. 5, no. 4, pp. 681–696, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7097022/

[57] H. Goudarzi and M. Pedram, "Hierarchical SLA-Driven Resource Management for Peak Power-Aware and Energy-Efficient Operation of a Cloud Datacenter," *IEEE Trans. on Cloud Computing*, vol. 4, no. 2, pp. 222–236, 2016.

[58] F. Hao, M. Kodialam, T. V. Lakshman, and S. Mukherjee, "Online allocation of virtual machines in a distributed cloud," *IEEE/ACM Trans. on Networking*, vol. 25, no. 1, pp. 238–249, Feb. 2017.

[59] A. C. Zhou, B. He, X. Cheng, and C. T. Lau, "A declarative optimization engine for resource provisioning of scientific workflows in geo-distributed clouds," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 647–661, March 2017.

[60] V. Persico, D. Grimaldi, A. Pescape, A. Salvi, and S. Santini, "A Fuzzy Approach Based on Heterogeneous Metrics for Scaling Out Public Clouds," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2117–2130, 2017.

[61] M. M. Nejad, L. Mashayekhy, and D. Grosu, "A family of truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," in *IEEE CLOUD 2013 Proceedings*, 2013.

[62] Z. Wang, M. M. Hayat, N. Ghani, and K. B. Shaban, "Optimizing cloud-service performance: Efficient resource provisioning via optimal workload allocation," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1689–1702, 2017.

[63] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing Resources Allocation for Virtualized Network Functions in a Cloud Center using Genetic Algorithms," *IEEE Trans. on Network and Service Management*, vol. 4537, no. 1, pp. 1–1, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7885521/

[64] D. Lago, E. Madeira, and D. Medhi, "Energy-Aware Virtual Machine Scheduling on Heterogeneous Bandwidths' Data Centers," *IEEE Trans. on Parallel and Distributed Systems*, vol. 29, no. 1, pp. 1–1, 2017.

[65] K. Salah, "A queueing model to achieve proper elasticity for cloud cluster jobs," in *IEEE CLOUD 2013 Proceedings*, 2013.

[66] W. Ellens, M. Zivkovic, J. Akkerboom, R. Litjens, and H. van den Berg, "Performance of cloud computing centers with multiple priority classes," in *IEEE CLUD 2012 Proceedings*, 2012.

[67] M. Hadji and D. Zeghlache, "Mathematical Programming Approach for Revenue Maximization in Cloud Federations," *IEEE Trans. on Cloud Computing*, vol. 5, no. 1, pp. 99–111, 2017.

[68] M. Tanaka and Y. Murakami, "Strategy-Proof Pricing for Cloud Service Composition," *IEEE Trans. on Cloud Computing*, vol. 4, no. 3, pp. 363–375, 2016.

[69] S. Zaman and D. Grosu, "An online mechanism for dynamic vm provisioning and allocation in clouds," in *IEEE CLOUD 2012 Proceedings*, 2012.

[70] Z. Huang and D. H. Tsang, "M-Convex VM Consolidation: Towards a Better VM Workload Consolidation," *IEEE Trans. on Cloud Computing*, vol. 4, no. 4, pp. 415–428, 2016.

[71] P. Kokkinos, T. Varvarigou, A. Kretsis, P. Soumplis, and E. Varvarigos, "Cost and utilization optimization of amazon ec2 instances," in *IEEE CLOUD 2013 Proceedings*, 2013.

APPENDIX

Solution	$\mathcal{T}_w$ 1	2	3	4	5
Oracle	0.000%	0.000%	0.000%	0.000%	0.000%
S-t Algorithm	1.065%	0.428%	0.336%	0.255%	0.208%
Heu1 (40%, 50%)	0.000%	0.000%	0.000%	0.000%	0.000%
Heu1 (50%, 60%)	0.000%	0.000%	0.000%	0.000%	0.000%
Heu1 (60%, 80%)	0.000%	0.000%	0.000%	0.000%	0.000%
Heu2 (0.9, 1.2)	13.530%	13.356%	13.252%	12.975%	13.090%
Heu2 (0.8, 1.3)	13.646%	13.600%	13.565%	13.461%	13.368%

TABLE 7
Response Time percentage violations $T_{slot} = 5$ min, spiky traffic and high noise.

Solution	$\mathcal{T}_w$ 1	2	3	4
Oracle	0.000%	0.000%	0.000%	0.000%
S-t Algorithm	1.065%	0.509%	0.289%	0.231%
Heu1 (40%, 50%)	0.000%	0.000%	0.000%	0.000%
Heu1 (50%, 60%)	0.000%	0.000%	0.000%	0.000%
Heu1 (60%, 80%)	0.000%	0.000%	0.000%	0.000%
Heu2 (0.9, 1.2)	13.530%	13.391%	13.287%	13.032%
Heu2 (0.8, 1.3)	13.646%	13.634%	13.565%	13.287%

TABLE 8
Response Time percentage violations for $T_{slot} = 5$ min, spiky traffic and low noise.

Solution	$\mathcal{T}_w$ 1	2	3
Oracle	0.000%	0.000%	0.000%
S-t Algorithm	4.259%	1.204%	0.625%
Heu1 (40%, 50%)	0.000%	0.000%	0.000%
Heu1 (50%, 60%)	0.000%	0.000%	0.000%
Heu1 (60%, 80%)	0.116%	0.208%	0.069%
Heu2 (0.9, 1.2)	15.509%	15.301%	13.681%
Heu2 (0.8, 1.3)	16.204%	15.602%	14.236%

TABLE 9
Response Time percentage violations for $T_{slot} = 10$ min, spiky traffic and high noise.

Solution	$\mathcal{T}_w$ 1	2	3	4
Oracle	0.000%	0.000%	0.000%	0.000%
S-t Algorithm	3.102%	0.694%	0.532%	0.370%
Heu1 (40%, 50%)	0.000%	0.000%	0.000%	0.000%
Heu1 (50%, 60%)	0.000%	0.000%	0.000%	0.000%
Heu1 (60%, 80%)	0.000%	0.000%	0.000%	0.000%
Heu2 (0.9, 1.2)	17.245%	17.083%	16.181%	16.042%
Heu2 (0.8, 1.3)	15.347%	15.440%	14.792%	14.560%

TABLE 10
Response Time percentage violations for $T_{slot} = 10$ min, spiky traffic and low noise.

	$\mathcal{T}_w = 1$		$\mathcal{T}_w = 2$		$\mathcal{T}_w = 3$		$\mathcal{T}_w = 4$		$\mathcal{T}_w = 5$	
Queue Length Q	Dropped Req. [%]	SLA Viol. [%]	Dropped Req. [%]	SLA Viol. [%]	Dropped Req. [%]	SLA Viol. [%]	Dropped Req. [%]	SLA Viol. [%]	Dropped Req. [%]	SLA Viol. [%]
Short timeout										
5	1.8676	0.00	1.5025	0.00	1.4378	0.00	1.2737	0.00	1.1881	0.00
10	0.9724	0.00	0.7626	0.00	0.7270	0.00	0.6314	0.00	0.5851	0.00
15	0.7707	0.00	0.5755	0.00	0.5464	0.00	0.4519	0.00	0.4048	0.00
20	0.6807	0.00	0.4927	0.00	0.4560	0.00	0.3683	0.00	0.3235	0.00
Long timeout										
15	4.2243	0.00	3.515	0.00	3.4030	0.00	3.0594	0.00	2.8769	0.00
20	2.0720	0.00	1.6784	0.00	1.6124	0.00	1.4288	0.00	1.3406	0.00
25	1.5284	0.00	1.2336	0.00	1.1889	0.00	1.0518	0.00	0.9905	0.00
30	1.3334	0.03	1.0633	0.07	1.0215	0.00	0.9003	0.00	0.8398	0.00
35	1.2485	0.00	0.9458	0.02	0.8667	0.00	0.8236	0.00	0.7600	0.00
40	1.1950	0.03	0.9001	0.01	0.8426	0.00	0.7737	0.07	0.7062	0.00

TABLE 11
Sensitivity to queue length and timeout