A Receding Horizon Approach for the Runtime Management of IaaS Cloud Systems

Danilo Ardagna*, Michele Ciavotta*, Riccardo Lancellotti†
*Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Milan, Italy
{danilo.ardagna, michele.ciavotta}@polimi.it
†Dipartimento di Ingegneria "Enzo Ferrari"
Università di Modena e Reggio Emilia , Modena, Italy
riccardo.lancellotti@unimore.it

***Abstract*—Cloud Computing is emerging as a major trend in ICT industry. However, as with any new technology it raises new major challenges and one of them concerns the resource provisioning. Indeed, modern Cloud applications deal with a dynamic context and have to constantly adapt themselves in order to meet Quality of Service (QoS) requirements. This situation calls for advanced solutions designed to dynamically provide cloud resource with the aim of guaranteeing the QoS levels. This work presents a capacity allocation algorithm whose goal is to minimize the total execution cost, while satisfying some constraints on the average response time of Cloud based applications. We propose a receding horizon control technique, which can be employed to handle multiple classes of requests. We compare our solution with an oracle with perfect knowledge of the future and with a well-known heuristic described in the literature. The experimental results demonstrate that our solution outperforms the existing heuristic producing results very close to the optimal ones. 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. Our analytical results are also validated through simulation, which shows also the impact on our solution of Cloud environment random perturbations.**

***Index Terms*—Auto-Scaling; Capacity Allocation; Optimization; QoS**

I. Introduction

The advent of Cloud Computing changed dramatically the Information and Communication Technology (ICT) industry over the past few years. Technology providers such as Google [22], Amazon [4], and Microsoft [28] aim at improving cost-effectiveness, reliability and computational power of cloud platforms, while an ever-growing population of firms reshape their business models to gain benefit from this new paradigm. The typical Cloud-based service exploits the joint effort of an *infrastructure provider*, that manages the cloud platform and is responsible for computational and networking capabilities, and a *service provider* that runs the actual application exploiting the resources of one or more infrastructure providers.

The growing popularity of Cloud Computing opens new challenges, especially in the area of resource provisioning. Guaranteeing adequate Quality of Service (QoS) requires management solutions that support performance prediction, monitoring of Service Level Agreements (SLA), and adaptive configuration, while satisfying the requirements of cost-effectiveness, reliability, and security. However, tools currently supplied by infrastructure providers to service providers, are often too basic and inadequate to suitably manage applications subject to highly variable workload conditions, with dynamic behavior characterized by uncertainty.

This open issue motivates the proposal of novel cloud management solutions that enable an application service provider, that we assume to follow a Software as a Service (SaaS) paradigm, to offer a set of applications that are deployed over an Infrastructure as a Service (IaaS) provider.

This paper fits within the scope of the MODAClouds project [11], [1] that aims at supporting model driven design and runtime management of Cloud applications. The main contribution of this paper is the proposal of a fast and effective capacity allocation technique that minimizes the execution costs of a Cloud application, guaranteeing QoS constraints expressed in terms of request average response time. To this aim, we introduce a receding horizon control strategy ompimising capacity allocation for a large number of request classes.

We compare the proposed algorithm with an *Oracle* characterized by a perfect knowledge of the future and with a well-known heuristic proposed in the literature [35], [39], [5].

The experiments demonstrate that our solution outperforms the considered heuristic producing high-quality results. When compared with the Oracle the cost gap is about 7% on average. With respect to heuristic solutions based on utilization thresholds [35], [39], [5], cost-savings range is $[50, 100]\%$.

Moreover, we analyze multiple solutions in terms of SLA violations using a simulated environment, where we consider also the time-varying performance degradation due to resource contention of Cloud applications. This analysis demonstrates that, even in the most demanding scenario, the proposed algorithm limits the number of SLA violations to less than 2% of the overall time slots.

Finally, an analysis with respect to the time scales used in the algorithm shows that a finer grained time scale, i.e., 5 minutes seems to have a positive effect on SLA violations. A similar positive effect is achieved also by increasing the

control time window used in our algorithm.

The remainder of this paper is organized as follows: In Section II and III we present our design assumptions and the Capacity Allocation (CA) problem formulation whereas our receding horizon algorithm is presented in Section IV. Section V is dedicated to assess the quality of our solution through experiments and numerical analysis. In Section VI we review other literature approaches. Conclusions are finally drawn in Section VII.

II. Problem Statement and Design Assumptions

In service-based systems, that are the focus of our study, an SLA contract is usually established between the SaaS provider and her end users for each Web Service (WS) application. For the sake of simplicity, we assume that WS applications are single tier and the underlying VMs are homogeneous in terms of their computing capacity (this assumption is far from being restrictive and it is compliant with auto-scaling mechanisms offered by current IaaS and PaaS solutions [6]; nowadays the management of heterogeneous VMs is relevant in private clouds, but it is not considered in public clouds [21], [13]). Furthermore, the type of VMs is fixed and selected at design time trough modern performance assessment tools, see [10].

In our approach we consider SLA based on the average response time and we assume that every WS application k has to provide an average request response time R_k lower than a threshold $\overline{R}_k$ (in the following the terms *WS application* k and *request class* k will be used interchangeably). As in [38], as a first approximation, we obtain R_k by modelling each WS application hosted within a VM as an M/G/1 queue in tandem with a delay center, and we assume that incoming requests are served according to the processor sharing scheduling policy, frequently used by Web service containers [25]. Furthermore, multiple VMs can run in parallel to support the same WS application. The delay center is used to model network delays and/or protocol delays introduced in establishing connections, etc. (see [38] for additional discussion).

We consider two types of VMs, *reserved* and *on-demand* and we denote by δ and ρ the costs for *on-demand* and *reserved* instances ($\rho < \delta$) respectively, charged on an hourly basis, while W indicates the overall number of available reserved instances.

Since we deal with time scales finer than one hour, we divide the time into *time slots* with a duration denoted by T_{slot}. We assume to pay for an instance as soon as it is required and, after that moment, we consider it as freely available for the next 60 minutes. At the beginning of the following charging hour, if we do not need that instance anymore, it is released; otherwise, it remains available and the SaaS provider will be charged again. The overall number of time slots in the charging interval (i.e., one hour) is denoted by n_c and it is assumed to be integer for simplicity. We denote as $\mathcal{T}_c = \{1, \ldots, n_c\}$ the set of charging time slots.

The optimization problem operates on a *observation window* of n_w time slots $\mathcal{T}_w = \{1, \ldots, n_w\}$. The decision variables of the problem are $(r_k^1, \ldots, r_k^{n_w})$ and $(d_k^1, \ldots, d_k^{n_w})$, i.e., the number of reserved and on demand VMs to be started during the observation window that, in conjunction with free instances of both types, have to serve the predicted incoming workload $(\widehat{\Lambda}_k^1, \ldots, \widehat{\Lambda}_k^{n_w})$. The final goal is to minimize the overall costs for reserved and on-demand VMs to serve the predicted arrival rate, while guaranteeing that the average response time of each WS application is lower than the SLA threshold. Our solution algorithm computes the values (r_k^t, d_k^t) for every time interval $t \in \mathcal{T}_w$, even if it starts or halts VMs only for the first time slot (r_k^1, d_k^1). This approach follows the so-called Receding Horizon Control paradigm [36]: The problem is solved for a finite horizon represented by the set $\mathcal{T}_w$, the decisions taken for the first time slot are actuated in the system accordingly and the optimization process is repeated considering the second time slot as a starting point.

Figure 1 graphically illustrates the relations among $\mathcal{T}_c$, $\mathcal{T}_w$ and T_{slot}. In this work we considered time slots of 5 and 10 minutes and observation windows with n_w from 1 up to 5 time slots, that is ranging from 5 to 50 minutes. As regarding the charging interval, we considered the common charging interval of one hour, i.e., it is composed of 6 to 12 time slots, respectively. In the problem formulation, we model the number of instances already paid and available for free at time slot t by means of parameters $\overline{r}_k^t$ and $\overline{d}_k^t$ for *reserved* and *on-demand* instances, respectively.

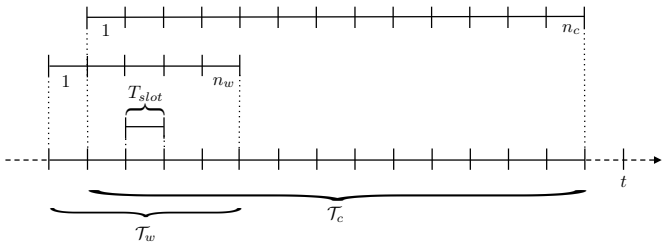

Fig. 1. Relations among $\mathcal{T}_c$, $\mathcal{T}_w$, and T_{slot}

Concerning the workload prediction, several methods have been adopted in many fields for decades [17] (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.

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

III. Optimization Problem Formulation

The Capacity Allocation (CA) problem is solved over an observation window $\mathcal{T}_w$ of n_w time slots and aims at minimiz-

System parameters

$\mathcal{K}$	Set of WS applications
δ	Time unit cost (measured in dollars) for *on-demand* VMs
ρ	Time unit cost (measured in dollars) for *reserved* VMs
$\mathcal{T}_w$	Set of time slots within the sliding time window
$\mathcal{T}_c$	Set of time slots within a 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 $\mathcal{T}_w$
$\overline{r}_k^t$	Number of *reserved* VMs freely available at time slot t in the interval under analysis, for request class k
$\overline{d}_k^t$	Number of *on-demand* VMs available for free at time slot t in the interval under analysis, for request class k
Λ_k^t	Real local arrival rate (measured in requests/sec) for request class k, at time slot t
$\widehat{\Lambda}_k^t$	Local arrival rate prediction (measured in requests/sec) for request class k, at time slot t
$\overline{R}_k$	Average response time threshold for request class k
W	Maximum number of *reserved* instances available

TABLE I
PARAMETERS OF THE CAPACITY ALLOCATION PROBLEM.

Decision Variables

d_k^t	Number of *on-demand* VMs to be allocated for request class k at time slot t
r_k^t	Number of *reserved* VMs to be allocated for request class k at time slot t

TABLE II
DECISION VARIABLES OF THE CAPACITY ALLOCATION PROBLEM.

ing the overall costs for *reserved* and *on-demand* instances to serve the arrival rate $\widehat{\Lambda}_k^t$, while guaranteeing SLA constraints.

The CA problem can be formulated as:

$$\text{(P)} \qquad \min_{r_k^t, d_k^t} \sum_{k\in\mathcal{K}} \left(\rho \sum_{t=1}^{n_w} r_k^t + \delta \sum_{t=1}^{n_w} d_k^t \right)$$

Subject to the conditions:

$$R_k(r_k^1, \overline{r}_k^1, \ldots, r_k^t, \overline{r}_k^t, d_k^1, \overline{d}_k^1, \ldots, d_k^t, \overline{d}_k^t, \widehat{\Lambda}_k^1, \ldots, \widehat{\Lambda}_k^t) \leq \overline{R}_k \quad \forall k \in \mathcal{K},\ \forall t \in \mathcal{T}_w \qquad (1)$$

$$\sum_{k\in\mathcal{K}} (r_k^t + \overline{r}_k^t) \leq W,\ \forall t \in \mathcal{T}_w \qquad (2)$$

$$r_k^t \geq 0,\ r_k^t \in \mathbb{N}\ \ \forall k \in \mathcal{K},\ \forall t \in \mathcal{T}_w$$

$$d_k^t \geq 0,\ d_k^t \in \mathbb{N}\ \ \forall k \in \mathcal{K},\ \forall t \in \mathcal{T}_w$$

Inequality (1) represents a set of non-functional constraints on the average response time of WS application k, where $R_k(\cdot)$ is function of the number of VMs active at every time instant t and the predicted workload.

Finally, inequality (2) represents a constraint on the overall number of available reserved VMs, which can not be greater than W (i.e., the number of VMs for which the SaaS subscribed a long term contract). Constraint (2) adds more complexity to the problem because it makes the problem no longer separable in $|\mathcal{K}|$ subproblems.

In [9], we formulated problem (P) as a Mixed Integer Linear Problem (MILP), which can be efficiently solved by commercial solvers.

IV. RECEDING HORIZON ALGORITHM

Figure 2 represents the basic schema of a controller implementing the receding horizon approach. At each time slot (marked by a clock element) the monitoring platform provides new workload predictions $\{\widehat{\Lambda}_k^1, \ldots, \widehat{\Lambda}_k^{n_w}\}$ for the current time window $\mathcal{T}_w$ and new estimates for the performance parameters. The optimizer component feeds the optimization model using the current application state expressed in terms of allocated VMs. Afterwards, the optimizer uses the model to calculate the most suitable number of VMs to allocate during the whole time window in order to guarantee the arranged SLAs. Finally, the optimizer operates on the running cloud application, through IaaS APIs, enacting only the first time slot of the attained allocation plan.

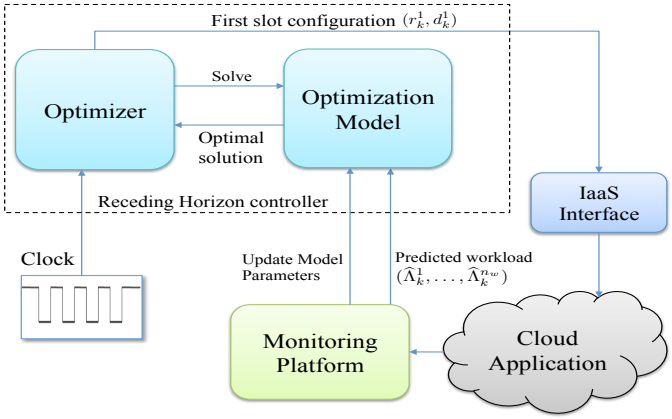

Fig. 2. Receding horizon controller.

Algorithm 1 is a high-level description of the operating elements of the receding horizon approach we propose with this work. The method can be roughly described by a sequence of four steps iteratively repeated to cover the overall time horizon. In the first step some model parameters (representing the state of the system and the predicted workload during the considered time window) are initialized at lines 2-7. In particular the system state is defined by the number of *reserved* $\overline{r}_k^t$ and *on-demand* $\overline{d}_k^t$ VMs available *free of charge* for each time slot of the observation window. Notice that, in order to manage the state of the system ($\overline{r}_k^t$, $\overline{d}_k^t$) in the execution of the algorithm for different time slots, we adopt the global parameters $N_{res,k}^t$, $N_{ond,k}^t$, which store the number of VM instances inherited from previous time slots and, therefore, available for free at time slot t. Afterwards, at line 8, we solve problem (P) to optimality by means of a third-party solver, whereas in line 10 the receding horizon paradigm is carried out by modifying the application deployment using only the values calculated for the first time slot of the considered time window, $Scale(k, r_k^1, d_k^1)$. Finally, the algorithm updates accordingly the system state, $N_{res,k}^{j+t}, N_{ond,k}^{j+t}$ (lines 11-14). It should be noticed that, since the VMs allocated at time t are available until the end of their charging period, the algorithm only update 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, otherwise we will be charged again and we will have them available for another charging period. Notice, though, new VMs are turned on only if strictly necessary (that is to handle increase of the workload). In other words, keeping old instances running is always preferred over switching on new ones. A more detailed description of this algorithm can be found in [9].

Algorithm 1 Receding Horizon Algorithm

```
1: procedure SOLUTION ALGORITHM
2:     for all k ∈ 𝒦 do
3:         for w ← 1, n_w do
4:             Λ̂_k^w ← GetPrediction(w, k)        } Initialization
5:             σ_k^w ← GetState(w, k)
6:         end for
7:     end for
8:     Solve (P, N_res, N_ond, Λ̂)  ←  Solving the current model
9:     for all k ∈ 𝒦 do
10:        Scale (k, r_k^1, d_k^1)  ←  Applying the changes according to the first time slot decisions
11:        for j ← 1, n_c do
12:            N_{res,k}^{t+j} ← N_{res,k}^{t+j} + r_k^1     } State update
13:            N_{ond,k}^{t+j} ← N_{ond,k}^{t+j} + d_k^1
14:        end for
15:    end for
16: end procedure
```

V. Experimental Results

In this Section we compare our approach with current state-of-art solutions implemented in modern Clouds and according to the auto-scaling policies, which can be implemented at IaaS provider level. In particular, our analysis focuses on the costs of the various solutions and their ability to satisfy response time constraints. Furthermore, some preliminary analysis demonstrates that our approach scales almost linearly with respect to the number of request classes. Systems up to 160 classes and 5 time slots can be solved in less than 200 sec. The complete scalability analysis is available in [9].

Section V-A presents the design of experiments. Section V-B analyzes the results achieved in terms of solution costs. Finally, Section V-C analyzes the impact of the control time scale resolution according to workload characteristics and SLAs fulfilment.

A. Design of experiments

The analyses performed in the following sections aim at representing real Cloud environments. We use a large set of randomly generated instances, obtained varying performance model parameters according to other literature approaches [3], [35], [39] or according to observations in real applications [7] [12]. The bound on the number of *reserved* instances is set to 10 in order to lead to saturation of the available reserved resources and, then, considering the worst case solutions where also on-demand resources are used (recall that relaxing constraint (2) problem (P) becomes much easier and can be separated into smaller problems, one for each WS application). For what concerns the cost parameters, we adopt the prices currently charged by IaaS Cloud Providers [4]. We decide to randomly generate instance costs, in order to replicate infrastructures spread worldwide. For on-demand instances the price ranges from 0.060 \$/h to 1.520 \$/h, while for reserved instances the cost is in the interval $[0.024, 0.608]$ \$/h.

Regarding the application workload Λ_k^t, we base our experiments on real measurements coming from a large popular website that wants to remain anonymous for privacy reasons. The workload exhibit a bi-modal distribution with two peaks around 10am and in the early afternoon, with low traffic in the early hours of the day and during the night. As in [27], [7], [8] and [2], we derive from the real workload a set of different workloads, one for each WS application k, by scaling the peak of each request class. Moreover, we add some noise as in [27], thus adapting original traces to a workload that captures the behavior of WS applications with different workload intensities. In our reference scenario, we consider a workload composed by 10 different classes of requests.

The workload prediction $\widehat{\Lambda}_k^t$ is obtained by adding white noise to each sample Λ_k^t, as in [7] to model prediction errors, [26]. The noise is proportional to workload intensity Λ_k^t, we considered two different level of noise *high* and *low*, see [7] for further details. Moreover, the prediction, in a real context, becomes less accurate while increasing the number of time slots in the future, for this reason the amount of noise is increased with the time step $t \in \mathcal{T}_w$ (further details are reported in the next Section). We decided to impose a white noise on the real workload, as in [2], because in this way the results are independent from the choice of the prediction technique.

B. Cost-Benefit Analysis

We perform a cost-benefit evaluation of our approach with a twofold aim: on one hand we compare the cost of our solution against another state-of-the-art technique. On the other hand, we assess the impact of the number of time slots of the sliding window on the CA solution.

As main performance indicator we use the overall virtual machines cost. The test case is based on a 24 hours time span and each hour is divided into time slots of 5 minutes.

As in [7] two workload models have been used: a *normal traffic model* where the number of client requests is characterized by a slow increase or decrease rate, and a *spiky traffic model*, that is the model used also in this paper. For space limitation here we report results achieved with the spiky workload, a more complete set of experiments can be found in [9]. Furthermore, we consider both a light level of noise, (corresponding to a more accurate prediction), and a heavy one. The amount of noise increases in the observation window, because the prediction gradually loses accuracy with $t \in \mathcal{T}_w$. The level of noise is also proportional to the workload sample Λ_k^j as reported in Table III. We consider a noise level up to

45%, since for larger values the prediction of the single sample becomes uncertain and comparable with a value equal to the sample itself, undermining the effectiveness of the overall approach.

t	Low noise	High noise
1	5%	20%
2	10%	25%
3	15%	30%
4	20%	40%
5	25%	45%

TABLE III
NOISE LEVELS ADOPTED.

The alternative solutions considered in our comparisons are the following:

- Our short-term algorithm (*S-t Algorithm*): the time scale T_{slot} has been set equal to 5 minutes and the observation window has been varied between 1 and 5 steps.
- Oracle-based algorithm (*Oracle*): has the same structure of our solution but the prediction of incoming traffic is 100% accurate, i.e., perfect knowledge of the future is assumed. This approach represents the theoretical perfect solution with the lowest cost and no SLA violations.
- Heuristic (*Heu*): that derives from [35], [39] and is currently implemented by some IaaS providers (see, e.g., Amazon AWS Elastic Beanstalk [5]). The heuristic implements an algorithm for auto-scaling the number of instances in order to handle workload fluctuations. As in our approach, the capacity allocation is performed over the overall time horizon considering an observation window $\mathcal{T}_w$ and employs the receding horizon approach by enacting only the decisions for the first time step in $\mathcal{T}_w$. The heuristic 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. 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 VMs utilization into the interval $(\overline{U}_1, \overline{U}_2)$. The thresholds used in the experiment are: $(\overline{U}_1, \overline{U}_2)$ = (40%, 50%), (50%, 60%), and (60%, 80%) as considered in [7].

Figures 3 and 4 present the mean solution cost evaluated over multiple test executions for a 24 hours time horizon. Specifically, we executed three different tests for each configuration of noise, and thresholds, for a total number of 36 runs; each run calculated the allocation for the overall time horizon defined for the configuration. For space limitations, in the present paper we report only a subset of the experiments performed. The complete analysis is available in [9].

From Figures 3 and 4 we observe two main results. The first is that the proposed solution is competitive with the Oracle: for both high and low noise levels the cost difference between the *S-t* algorithm and the Oracle is at most in the order of tens of dollars. On the other hand, the Heuristic always provides the worst solutions with costs more than double with respect to our proposal. A different choice for the utilization thresholds may improve the performance of the Heuristic, but this solution remains significantly more expensive when compare to the *S-t* alternative.

The second main result concerns the impact of $\mathcal{T}_w$ on costs. For both workloads and for every algorithm we observe a general increase in costs as the observation window grows. This result can be explained by the combination of two effects: on one hand increasing the observation window leads to a more conservative behavior of the algorithms that strive to optimize the allocation problem over longer periods of time. On the other hand, traffic prediction error tends to negatively affect the quality of solutions, to the point where increasing the observation window simply forces the optimization of a problem that is not related to the actual future traffic patterns.

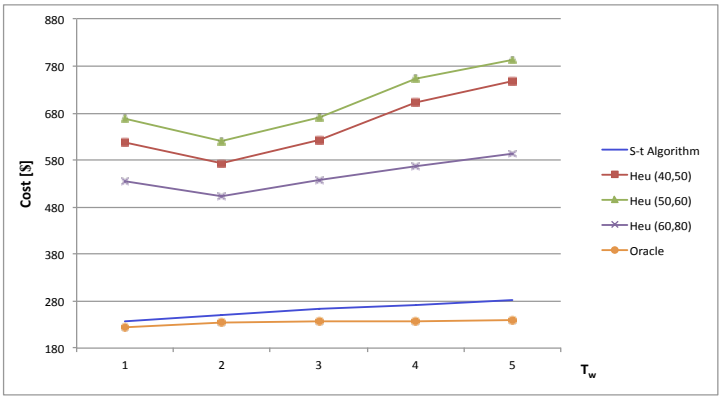

Fig. 3. Solution cost, $T_{slot} = 5min$, low noise level.

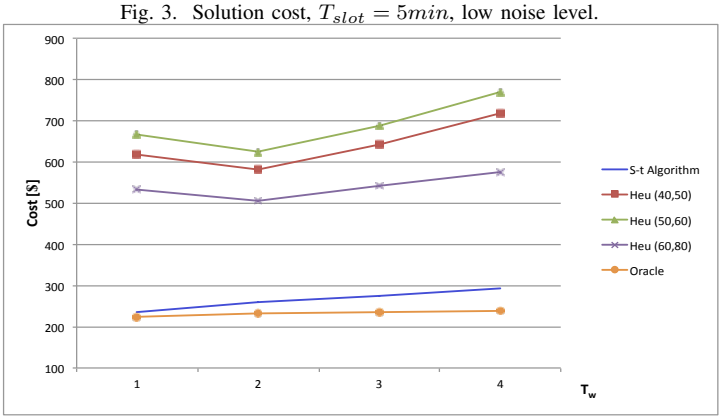

Fig. 4. Solution cost, $T_{slot} = 5min$, high noise level.

C. Time scale Analysis

We now evaluate the impact of time scale on the proposed receding horizon algorithm to understand which is the best time granularity. Analyses have been supported by a discrete event simulator based on the Omnet++ framework [30] that has been developed ad hoc for this purpose. Our simulator is able to capture the time-varying performance degradation due to resource contention of Cloud applications by introducing Random Environments (REs) [16].

REs are useful abstractions to represent variability in a system behavior. 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 [16], [15] and have been successfully used also for analyzing the performance of servers adopting dynamic voltage-scaling technologies that change over time CPU frequency to reduce energy consumption [20].

As discussed in Section II, we model each VM serving a generic WS application k as a M/G/1 processor sharing queue in tandem with a delay center. An RE with two stages is used to model the variability of performance of a virtual hardware due to resource contention (see Figure 5). 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 $D_k^{fast} < D_k^{slow}$. We consider the transition probability between the two stages (p^{fast} and p^{slow}) such that the average time for fast-to-slow and slow-to-fast transitions is 415s and 40s, respectively. Performance degradation between the two stages is modeled by setting $\mu_k^{fast} = \alpha\mu_k^{slow}$, $D_k^{fast} = D_k^{slow}/\alpha$ with $\alpha = 10$ (note that problem (P) solution adopts μ_k^{fast} D_k^{fast} values). All these parameters are obtained from a preliminary analysis on a real Cloud system (see Figure 6) and according to values reported in the literature [16], [15]. The ability of our simulator to capture the inherent variability of the cloud environment is evident from a comparison of Figure 6 representing the response time of a SPECWeb2005 benchmark deployed over Amazon EC2, and Figure 7 that represents a trace obtained from our simulator.

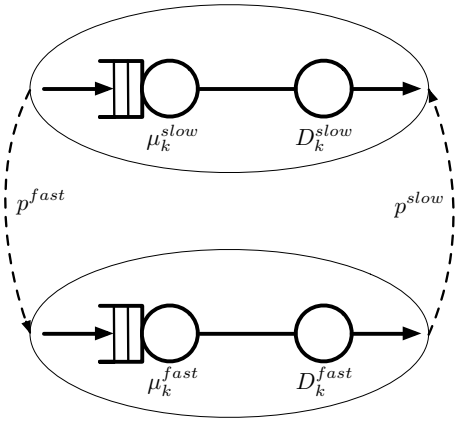

Fig. 5. Random Environment modeling Cloud resource contention.

The inter-arrival time of requests is modeled separately for each application k using a time-variant Gaussian model, where for each time interval t, we change the request inter-arrival time $\frac{1}{\Lambda_k^t}$. A round-robin request dispatcher distributes requests over the servers associated to the WS application k.

The time for request processing is modeled according to a lognormal distributions, as in [3] for both servers and delay centers. The standard deviation of lognormal distributions range from two to ten times their mean, as observed for several real applications [34], [31]. Furthermore, an upper bound for service and delay times has been set equal to 5 times the mean to limit the effect of the heavy tailed distribution used in our model. Furthermore, to cope with the huge build-up in queue length due to temporary congestion, we model our server queue as a finite capacity queue. In our experiments we consider a queue length equal to 10 and 40 elements for $T_{slot} = 5$ and $T_{slot} = 10$ minutes, respectively. A more detailed sensitivity analysis with respect to the queue length is left as an open issue to be addressed by future work.

For all the experiments, we simulate the system for 24 hours considering T_{slot} equal to 5 and 10 minutes. For each time slot the *S-t* algorithm with a prediction characterized by low noise is used to address the capacity allocation problem. Simulation results refer to data averaged over 10 runs of the simulator, to guarantee the reliability of the results.

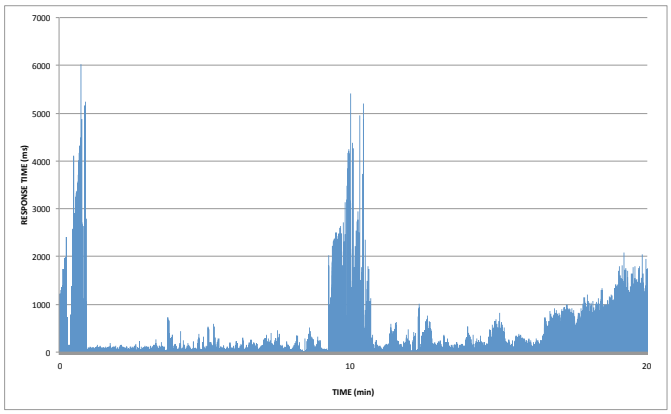

Fig. 6. Real Trace.

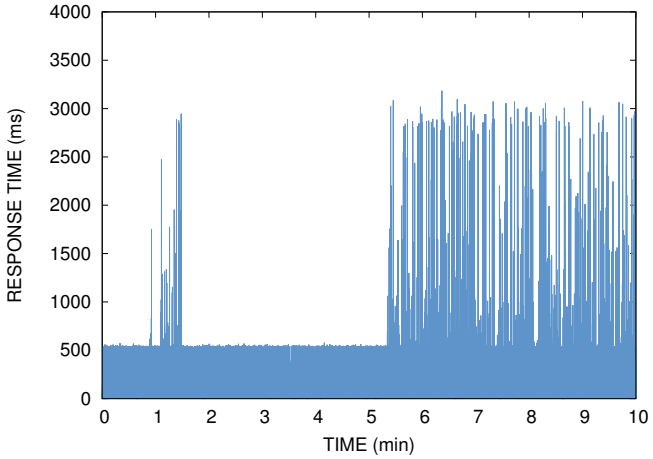

Fig. 7. Simulation Output.

The main performance indicators considered in these experiments are the percentage of time slots where the average response time exceeds the SLA threshold (SLA violations columns in Table IV) and the percentage of requests dropped as a result of the finite queue length (Dropped requests columns). A further representation of the simulation output compares the average response time over 24 hours with the SLA, showing the violations (see Figure 8).

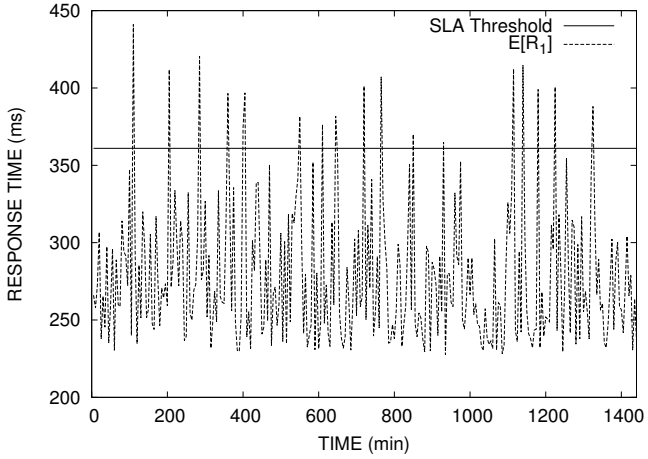

Fig. 8. Average response time vs. SLA threshold.

$\mathcal{T}_w$	SLA Violations [%]		Dropped Requests [%]	
	5 min	10 min	5 min	10 min
1	0.49	1.74	5.56	6.71
2	1.08	0.56	5.91	6.34
3	0.90	1.81	5.99	6.26
4	1.15	1.88	5.61	5.95
5	0.00	0.00	5.56	3.19

TABLE IV
SIMULATION RESULTS

Table IV provides two clear messages. First, a control time granularity of 5 minutes tends to provide better performance if compared to granularity of 10 minutes both in terms of SLA violations and in terms of dropped requests. To this aim it is worth to note that the fraction of time slots with SLA violations is quite limited, with worst cases of less than 1.2% and less than 2% for T_{slot} of 5 and 10 minutes, respectively. A second result is that as the observation window grows to a length of 5, no SLA violations occurs. This result can be explained by reminding that as $\mathcal{T}_w$ increases, the algorithm becomes more conservative, meaning that, if the impact of noise is not disruptive, the CA problem obtains better solutions. A similar trend can be observed for the percentage of dropped requests, that decreases as $\mathcal{T}_w$ grows.

This simulation-based evaluation of the proposed algorithm is a first step towards a better understanding of the algorithm performance. A complete sensitivity analysis to noise level, parameters of REs, duration of T_{slot}, and length of $\mathcal{T}_w$ deserves additional study to better understand the correlation between these parameters. However, such comprehensive analysis is left as part of future works.

VI. Related Work

Several approaches have been proposed to manage efficiently the resources of Cloud systems. Since Cloud computing is a promising technology, rapidly growing and appealing for industries, there is a multitude of studies in different research fields.

Many studies concern the resource allocation problem, with constraints on cost and execution time. The work presented in [18] helps to define a realistic SLA with customers and support a dynamic capacity allocation able to adapt to workload fluctuations. The model formulated represents a Cloud center with a $M/M/C/C$ queuing system, with different priority classes. Authors in [23] show a solution method for a multi-dimensional resource allocation problem in data-centers, while guaranteeing SLAs for clients with applications that require multiple tiers of service to be executed. The work in [29] proposes a VM provisioning problem from the IaaS provider perspective, where cloud customers bids for the use of VM resources and have no incentives to lie about their requested bundles. The problem is formulated as an integer program and solved by greedy heuristics. In [2] the VM placement problem for a PaaS is solved at multiple time scales through a hierarchical optimization framework. [24] aims at optimizing the cost and the utilization of a set of applications running on Amazon EC2 proposing a vertical auto-scaling mechanism, i.e., the algorithm, given the current set of instances used (their number, type, utilization), proposes a new set of instances for serving the same load, so as to minimize cost and maximize utilization, or increase performance efficiency.

A methodology to integrate workload burstiness in performance models is proposed in [14]. The authors exhibit a parametrized queuing model that can be used to predict performance in systems even in the challenging case where there are bottleneck switches among the various servers. In [19] is presented a strategy for Cloud resource allocation that, compared with traditional approaches related on cost and performance, ensures predictable processing speed and SLAs.

The work in [32] provides a queuing model and closed-form solutions for estimating Cloud applications response time, blocking probability, and throughput and proposes a very fast solution for estimating the minimum number of VMs required to achieve performance objectives.

Finally, an online VM provisioning method is proposed in [37]. The solution solves allocation problems with partial information, calculating allocation and revenues as customers arrive at the system and place their requests. Most of the research studies deal with a single step prediction of the Cloud system workload. This paper is one of the first contributions proposing and analyzing the effectiveness of receding horizon solutions. The work in [33] is the closest contribution to our approach where a predictive controller for key-value storage systems is presented. The controller builds, according to the system current state, the optimal set of actions (e.g., move or copy data among multiple instances) and executes the first action of the sequence and then recomputes the optimal sequence again. However, the impact of the workload prediction error and controller time scales are not analyzed.

VII. Conclusions

In this work we proposed a capacity allocation technique able to minimize the execution cost of Cloud applications providing multiple classes SLA guarantees. An extensive analysis using both analytical techniques and simulation demonstrates that our approach is a viable solution that 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 is $[50, 80]\%$.

Experiments carried out with a simulator demonstrate that the considered solution limits the number of SLA violations to less than 2% of the total number of time slots. Furthermore, a preliminary sensitivity analysis with respect to the time scale used in the algorithm suggests that long time windows with a fine grained subdivision of time slots improves performance in terms of SLA violations.

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 condition. Finally, experiments in a real prototype environment will be also performed.

ACKNOWLEDGEMENTS

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

REFERENCES

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

[2] 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 Transactions on Dependable and Secure Computing*, 10(5):253–272, 2013.

[3] J. Almeida, V. Almeida, D. Ardagna, I. Cunha, C. Francalanci, and M. Trubian. Joint admission control and resource allocation in virtualized servers. *Journal of Parallel and Distributed Computing*, 70(4):344 – 362, 2010.

[4] Amazon Inc. Amazon Web Services. http://aws.amazon.com/.

[5] Amazon Inc. AWS Elastic Beanstalk. http://aws.amazon.com/elasticbeanstalk/.

[6] Amazon Inc. Elastic Load Balancing. http://aws.amazon.com/elasticloadbalancing/.

[7] D. Ardagna, S. Casolari, M. Colajanni, and B. Panicucci. Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems. *Journal of Parallel and Distributed Computing*, 72(6):796 – 808, 2012.

[8] D. Ardagna, S. Casolari, and B. Panicucci. Flexible distributed capacity allocation and load redirect algorithms for cloud systems. In *IEEE Cloud 2011*, 2011.

[9] 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, 2014.

[10] D. Ardagna, M. Ciavotta, M. Miglierina, M. Shokrolahi, G. P. Gibilisco, G. Casale, and J. F. Pérez. MODACloudML QoS abstractions and prediction models specification - initial version. Modaclouds eu project deliverable, 2013.

[11] 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 ICSE Workshop*, pages 50 –56, june 2012.

[12] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Trans. Services Computing*, 5(1):2–19, 2012.

[13] C. Canali and R. Lancellotti. Exploiting ensemble techniques for automatic virtual machine clustering in cloud systems. *Autom. Softw. Eng.*, pages 319–344, 2014.

[14] G. Casale, N. Mi, L. Cherkasova, and E. Smirni. Dealing with burstiness in multi-tier applications: Models and their parameterization. *Software Engineering, IEEE Transactions on*, 38(5):1040 –1053, sept.-oct. 2012.

[15] G. Casale and M. Tribastone. Fluid analysis of queueing in two-stage random environments. In *QEST*, pages 21–30, 2011.

[16] G. Casale and M. Tribastone. Modelling exogenous variability in cloud deployments. *SIGMETRICS Perform. Eval. Rev.*, 40(4):73–82, Apr. 2013.

[17] S. Casolari and M. Colajanni. Short-term prediction models for server management in internet-based contexts. *Decis. Support Syst.*, 48(1):212–223, Dec. 2009.

[18] W. Ellens, M. Zivkovic, J. Akkerboom, R. Litjens, and H. van den Berg. Performance of cloud computing centers with multiple priority classes. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 245 –252, june 2012.

[19] M. Ferber, T. Rauber, M. Torres, and T. Holvoet. Resource allocation for cloud-assisted mobile applications. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 400 –407, june 2012.

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

[21] D. Gmach, J. Rolia, and L. Cherkasova. Selling t-shirts and time shares in the cloud. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 539–546, May 2012.

[22] Google Inc. Google inc.

[23] H. Goudarzi and M. Pedram. Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 324 –331, july 2011.

[24] P. Kokkinos, T. Varvarigou, A. Kretsis, P. Soumplis, and E. Varvarigos. Cost and utilization optimization of amazon ec2 instances. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 518–525, June 2013.

[25] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan. Loosely coupled coordinated management in virtualized data centers. *Cluster Computing*, 14(3):259–274, 2011.

[26] D. Kusic, N. Kandasamy, and G. Jiang. Approximation modeling for the online performance management of distributed computing systems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 38(5):1221 –1233, oct. 2008.

[27] 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*, 12(1):1–15, 2009.

[28] Microsoft. Microsoft corporation. http://www.microsoft.com/.

[29] M. M. Nejad, L. Mashayekhy, and D. Grosu. A family of truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds. In *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, CLOUD '13, pages 188–195, Washington, DC, USA, 2013. IEEE Computer Society.

[30] OMNeT++ Discrete Event Simulation System, 2014. – http://www.omnetpp.org.

[31] V. Paxson and S. Floyd. Wide area traffic: The failure of poisson modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, June 1995.

[32] K. Salah. A queueing model to achieve proper elasticity for cloud cluster jobs. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 755–761, June 2013.

[33] B. Trushkowsky, P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. The scads director: Scaling a distributed storage system under stringent performance requirements. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

[34] E. Veloso, V. Almeida, W. Meira, Jr., A. Bestavros, and S. Jin. A hierarchical characterization of a live streaming media workload. *IEEE/ACM Trans. Netw.*, 14(1):133–146, Feb. 2006.

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

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

[37] S. Zaman and D. Grosu. An online mechanism for dynamic vm provisioning and allocation in clouds. In *IEEE Cloud 2012*, pages 253 –260, june 2012.

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

[39] 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. *Journal of Cluster Computing*, 12(1):45–57, 2009.