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

Danilo Ardagna, Michele Ciavotta, Riccardo Lancellotti, Michele Guerriero

October 8, 2018

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. 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 Cloud based applications. We propose a receding horizon control technique, which can be employed to handle multiple classes of requests. An extensive evaluation of our solution against an Oracle with perfect knowledge of the future and well-known heuristics presented 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 we violated QoS constraints for only 8 minutes over a day versus up to 260 minutes of other approaches). 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 validated through simulation, which shows also the impact on our solution of Cloud environment random perturbations. Finally, experiments on a prototype environment demonstrate the effectiveness of our approach under real workloads.

Index terms— Auto-Scaling, Capacity Allocation, 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 players of the ICT industry (e.g., Google [1], Amazon [2], and Microsoft [3]) aim to improve

cost-effectiveness, reliability and the overall computing power consumption of Cloud systems. This effort is shifting the business models of many companies that try to gain benefit from this new paradigm. The delivery at large scale of services through a Cloud computing platform requires a collaboration between one or more *infrastructure providers*, that manage the Cloud platform and are responsible for providing 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. In particular, we need to guarantee an adequate Quality of Service (QoS) for the Cloud customers. This objective, in turn means that we need management solutions that support performance prediction, monitoring of Service Level Agreements (SLAs), and adaptive configuration, while satisfying the requirements of cost-effectiveness, reliability, and security. Current solutions aimed at enforcing SLA guarantee are mainly focused at single Clouds. If we consider a multi-Cloud solution that is the case where a Cloud provider manages multiple data centers or where we have multiple providers each with his own data center, we have few solutions (e.g., the Amazon autoscaling rules[1]) which follow a purely reactive approach: Scaling is triggered when exceeding a threshold on some monitoring metric (e.g., CPU utilization is above 60%). However, providing SLA guarantees with a multi-Cloud scope is fundamental when we want to meet also the availability requirements of mission critical services.

We propose novel algorithms for the delivery of services that can take full advantage from the Cloud characteristics such as the ability to dynamically manage requests in a scenario characterized by multiple Cloud providers distributed geographically, each hosting several applications.

Our proposal follows a dual time-scale approach, that has been proved successful in literature [4]. At a coarse-grained time scale, we propose an innovative technique for the distribution of requests among multiple Cloud providers while at the fine-grained time scale we apply a receding horizon algorithm to allocate VMs in each data center meeting the future workload demands that is an evolution of [5]. Our effort is not limited to the algorithmic proposal, but we provide three types of validation for our technique that represent additional contributions of this paper. First, we provide an analytical framework to compare the proposed solution with an *Oracle* with perfect knowledge of the future and with well-known heuristics proposed in the literature [6–8] based on utilization thresholds. Second, we develop a new simulator that captures the data center behavior including the presence of exogenous effect on the overall performance, modeled through *Random Environments*, as proposed in [9]. This simulator allows us to evaluate the SLA violations of our proposal in realistic situations. Furthermore, we use the simulator for a thorough sensibility analysis with respect to the time granularity of the control strategy, the random environments parameters and the workload scenarios. Third, we present a first prototype of system, developed within the scope of the MODAClouds project [10, 11] and we demonstrate the viability of the proposed algorithms.

[1] http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/policy_creating.html

Our results confirm that our solution outperforms the heuristics producing results very close to the optimal ones (cost savings range is [30, 80]%), and reducing the number of QoS violations (in the worst case we violated QoS constraints for only 8 minutes over a day versus up to 260 minutes of other approaches). 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. Simulation shows that our solution is robust to Cloud environment random perturbations. Finally, the results achieved in the prototype environment demonstrated that the number of SLA violation under a real workload are less than 2%.

The remainder of this paper is organized as follows: In Section 2 we present the global problem of managing the delivery of web-based services over multiple Cloud infrastructures. Section 3 provides the theoretical definition of the fine- and coarse-grained part of the problem. Section 4 describes the algorithms used for both the request distribution over multiple data centers and for the VM allocation at the level of single data center. Section 5 evaluates the quality of our solution through experiments and simulation. In Section 6 we review other literature approaches. Finally, Section 7 contains some concluding remarks.

2 Problem Statement and Assumptions

Here we introduce the system model and the design assumptions used as reference throughout the paper. In particular, we propose a hierarchical modeling of the problem with two different temporal granularities, namely *long-term* and *short-term*.

2.1 Problem overview

In this paper we assume the point of view of a Software as a Service (SaaS) provider that deploys a suite of applications, in form of Web Services (WS), on multiple Infrastructure-as-a-Service (IaaS) providers. The applications are heterogeneous in terms of resource demands, workload profiles they are subject to, and SLA requirements. Services with different SLAs and workloads are categorized into independent classes.

Figure 1 depicts the reference 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 (in the following the terms *WS application* and *request class* will be used interchangeably). For each class $k \in \mathcal{K}$ the input workload is referred to as Λ_k. 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, we assume that each VM can only host a single web service application. Services can be replicated on multiple Clouds or on multiple data centers of the same provider (e.g., regions or availability zones in the Amazon EC2 platform [12]) within the same Cloud.

This approach is possible thanks to a software layer developed within MODAClouds project [10, 11], which allows concurrent execution, runtime migration and data synchronization of applications among multiple Cloud providers. MODAClouds provides full-stack solutions for the modeling, deployment, and runtime management

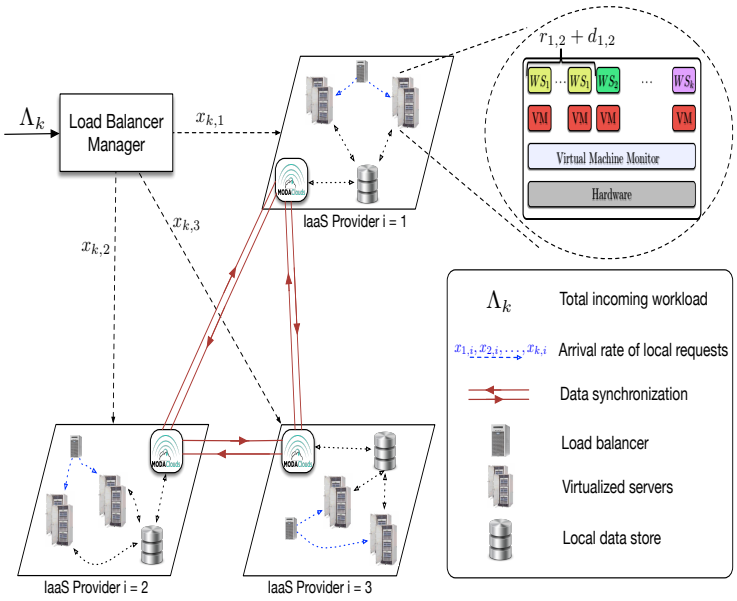

Figure 1: Cloud infrastructures and data migration and synchronization system.

of multi-Cloud applications. It aims at removing all limitations and technological lock-ins that presently prevent the execution of applications on different Clouds. It has been demonstrated that deploying applications on multiple Clouds provides benefits in terms of availability and implies cost savings for SaaS providers, due to IaaS competition and workload redistribution at runtime [13]. From a technological perspective, running applications over multiple Clouds requires a reliable mechanism of data synchronization among database replica (either relational or NoSQL) hosted on different Clouds. To this end, MODAClouds runtime platform also provides a distributed middleware (see Figure 1) in charge of synchronizing data among (even technologically) different databases [14].

We assume that, within a single data center, multiple homogeneous VMs implementing the same WS class can run in parallel, evenly sharing the incoming workload. This corresponds to the solution currently implemented by IaaS providers (e.g., [15]).

We assume that an SLA contract associated with each WS class $k \in \mathcal{K}$ is established between the SaaS provider and his customers. This contract specifies the QoS levels expressed in terms of average response time R_k for each class k that the SaaS provider pledges to meet.

In this work we aim to address the problem of minimizing SaaS leasing costs by solving jointly the multi-Cloud Capacity Allocation (CA) and Load Sharing (LS) problems. We propose a *divide et impera* optimization approach that considers two different time scales, that we refer to as *long-term* and *short-term*.

At long-term, we decide about the Load Sharing: in other words at this stage we

decide how to distribute the expected workload (for the next hour) among different IaaS providers with the objective of minimizing the compound VM leasing costs. The long-term algorithm splits, for each application k, a prediction for the global incoming request flows $\tilde{\Lambda}_k$[2] 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 we make this analysis on hourly basis planning the workload allocation one hour in advance [16, 17]. It is worth noticing that in order to calculate the optimized workload shares for each provider the long-term problem considers both LS and CA. The fine-grained problem is a local (for each Cloud $i \in \mathcal{I}$) CA optimization, with a time-scale in the order of 10 or 5 minutes, considering as incoming workload the load distribution $x_{k,i}$ calculated by the long-term problem. The objective is to determine the number of VMs able to serve the local incoming workload, while minimizing costs and guaranteeing that the average response time is below a given threshold $\overline{R}_k$, i.e., $R_k \leq \overline{R}_k$.

Even if IaaS providers usually charge software providers on an hourly basis for every instance, in this work we want to analyze the best time scale of the short-term problem, between 5 or 10 minutes, for the CA problem. By considering two different time scales, we are able to capture two phenomena related to Cloud applications. The fine-grain workload traces exhibit a high variability due to the short-term changes of the typical web-based workload. For this reason, the fine-grain time scale provides the allocation of new instances with the purpose of avoiding the achievement of saturation conditions. In the coarse-grain time scale, the workload traces are more smoothed and not characterized by the instantaneous peaks typical of the fine-grain time scale. These characteristics allows us to use the long-time scale algorithm to predict the workload trend that represents useful information for the capacity allocation algorithm.

The long-term problem has a time horizon denoted by T_{long} that represents the number of hours on which we calculate the workload allocation among multiple providers. The short-term problem operates on time slots of few minutes, and we denote its time granularity as T_{slot}.

2.2 Long-term request distribution mechanism

The long-term problem concerns the distribution of the incoming requests load $\Lambda_k, k \in \mathcal{K}$ over 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 so that the outcome of the long term problem is used as an input of the short term problem at the beginning of the next hour. To predict the per-class workload, multiple solutions have been proposed, ranging from the rule of thumb of simply considering the accesses observed in previous days at the same hour of the day up to more complex methods involving regression-based techniques [16, 18]. To distribute the requests among the providers, we model each WS application hosted within a Virtual Machine (VM) as an M/G/1 queue in tandem with a delay center as in [19] and we assume that incoming requests are served according to the processor sharing scheduling policy, frequently

[2]In the following $\tilde{\cdot}$ 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.

used by web service containers [20]. The delay center $D_{k,i}$ is used to model network delays and/or protocol delays introduced in establishing connections, etc. Furthermore, we assume that multiple VMs can run in parallel to support the same WS application (see Figure 2).

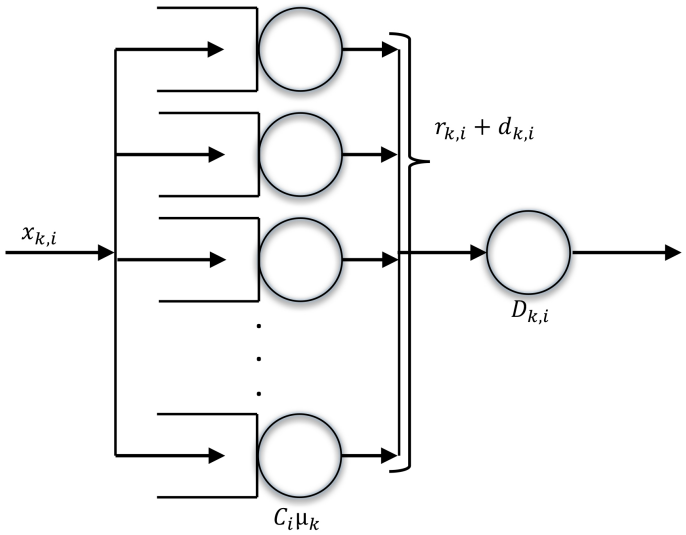

Figure 2: System performance model

As previously stated, we consider that a SLA contract, in the form of a threshold $\overline{R}_k$ on the average response time, is in force. We translate this bound as a SLA for every provider such that $R_{k,i} \leq \overline{R}_k$ for every WS application k and provider i. Within each provider i, we assume that VMs are homogeneous in terms of their computing capacity C_i (this assumption is not restrictive and compliant with auto-scaling mechanisms offered by current IaaS and PaaS solutions [15]). As far as costs are concerned, we assume two pricing models are offered by every provider: they supply *reserved* and *on-demand* VMs. We denote by δ_i and ρ_i the hourly fees for on-demand and reserved instances ($\rho_i < \delta_i$), respectively. Let W_i indicates the overall number of available reserved instances. For the service rate, we consider μ_k as the maximum service rate for serving a class k request with a VM of capacity 1.

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 $x_{k,i}$.

2.3 Short-term receding horizon control mechanism

The short-term problem is managed at the local level of a single Cloud provider. The data center is modeled in the same way as the long-term problem (see Fig. 2). We assume that the performance parameters are continuously updated at runtime (see [19]

for further details) in order to capture transient behaviors, VMs network, I/O interference [21], and time-dependent performance of Cloud resources [22, 23]. To model the time in the short-term problem, we consider multiple time slots of duration T_{slot}. Within the time horizon of the long-term problem, we focus on a sliding window of T_w future time slots. Since we deal with time scales finer than one hour, we assume to pay for an instance as soon as it is required and after that moment we consider it as freely available, until the end of its lease term. At the beginning of the following hour, if we do not need that instance anymore it is released; otherwise, it remains available and the fee is charged again. 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, we model the number of instances already paid and available for free 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 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 during the observation window $\mathcal{T}_w = \{1, \ldots, n_w\}$ that, in conjunction with free instances of both types, 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 [24], where the optimal solution achieved considering the whole time window, but the algorithm enforces only the decisions calculated for the nearest time step. 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 VMs only at the first time slot, according to $(r^1_{k,i}, d^1_{k,i})$. The optimization process is then repeated considering the second time slot as the 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 we considered time slots of 5 and 10 minutes and 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, we considered the common VM lease term of one hour, i.e. it is composed of 6 or 12 time slots.

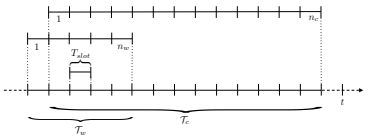

Figure 3: Relationships between $\mathcal{T}_c$, $\mathcal{T}_w$ and T_{slot} over time

Concerning the workload prediction, several methods have been adopted over the last decades [18] (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 1 and 2.

3 Optimization problems formulation

This section provides the mathematical formulation of the coarse- (Section 3.1) and fine-grained (Section 3.2) optimization problems.

3.1 Long-term problem

The idea of this problem is to split the workload prediction $\tilde{\Lambda}_k$ between different IaaS, in order to minimize the cost for the VMs. The problem is solved hourly, with a scope limited to the next hour.

The average response time for class k at provider i is given by:

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

which it follows the presence of the queue center and the delay center. According to M/G/1 equilibrium condition, it must be:

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

that we write as:

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

So, we can write the formula of the average response time as:

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

subject to the QoS condition:

$$R_{k,i} \leq \overline{R}_k$$

After some algebra we get:

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

and then:

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

because $1 - \left(\overline{R}_k - D_{k,i}\right) C_i\mu_k < 0$. Indeed, $\overline{R}_k > D_{k,i}$ according to the QoS condition described before, $C_i > 0$ and $\mu_k > 0$ in agreement with system properties.

Hence, if we denote with δ_i the cost of *on-demand* VM instances and with ρ_i the cost of the *reserved* instances for provider i, the joint Capacity Allocation and Load Sharing problem can be formulated as:

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

Subject to the conditions:

$$r_{k,i} + d_{k,i} - \frac{x_{k,i}}{C_i \mu_k} > 0 \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{2}$$

$$r_{k,i} + d_{k,i} + x_{k,i} \left(\frac{\overline{R}_k - D_{k,i}}{1 - \left(\overline{R}_k - D_{k,i}\right) C_i \mu_k} \right) \geq 0 \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{3}$$

$$\sum_{i \in \mathcal{I}} x_{k,i} = \tilde{\Lambda}_k \qquad \forall k \in \mathcal{K} \tag{4}$$

$$x_{k,i} \geq \gamma_i \tilde{\Lambda}_k \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{5}$$

$$\sum_{k \in \mathcal{K}} r_{k,i} \leq W_i \qquad \forall i \in \mathcal{I} \tag{6}$$

$$r_{k,i} \geq 0,\ r_{k,i} \in \mathbb{N} \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{7}$$

$$d_{k,i} \geq 0,\ d_{k,i} \in \mathbb{N} \qquad \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{8}$$

As we wrote before, conditions (2) represents the equilibrium condition of the adopted queue model M/G/1. This constraint has a relation of dependence with constraint (3), which concerns the response time bound defined in the service level agreement. The equilibrium condition (2) satisfies the condition of existence of the response time formula (1), because it assures that the denominator has to be different from 0.

Moreover, we can see that if the response time denominator is close to 0, $R_{k,i}$ becomes very high, especially greater than the SLA threshold $\overline{R}_k$. So, if the condition (3) is satisfied, also the (2) is fulfilled.

Constraints (4) ensure that the traffic assigned to individual providers equals the overall load predicted for class k jobs. In this way we ensure that the whole incoming traffic will be managed.

Constraints (5) guarantee that every IaaS obtains at least a fraction γ_i of workload, avoiding situations where all the workload is simply forwarded to the minimum-cost provider.

The constraint (6), increases problem complexity with the creation of a relation between different classes of request: At the same provider it is allowed to allocate the maximum number W_i of *reserved* instances, considering all the applications deployed; without this constraint, we could resolve the problem for each class request individually.

(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.

As a results of the problem, the workload of the system is redirected to each provider at every instant according to the probability defined by $\frac{x_{k,i}}{\sum_{i' \in \mathcal{I}} x_{k,i'}}$.

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 predicted arrival rate is not the overall arrival rate from the long-term problem, but is referred to just the time slot t. 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:

$$\sum_{\tau=1}^{t} \left(r^\tau_{k,i} + d^\tau_{k,i} \right) + \overline{r}^t_{k,i} + \overline{d}^t_{k,i} > \frac{\tilde{x}^t_{k,i}}{C_i \mu_k}, \qquad \forall k \in \mathcal{K}, \forall t \in \mathcal{T}_w \quad (9)$$

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

$$\sum_{k \in \mathcal{K}} (r^t_{k,i} + \overline{r}^t_{k,i}) \leq W_i \qquad \forall t \in \mathcal{T}_w \quad (11)$$

$$r^t_{k,i} \geq 0, \; r^t_{k,i} \in \mathbb{N} \qquad \forall k \in \mathcal{K} \; \forall t \in \mathcal{T}_w \quad (12)$$

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

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

It is worth to be note that inequality (9) 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:

$$R^t_{k,i} = \frac{1}{C_i \mu_k - \frac{\tilde{x}^t_{k,i}}{\overline{r}^t_{k,i} + \overline{d}^t_{k,i} + \sum_{\tau=1}^{t} (r^\tau_{k,i} + d^\tau_{k,i})}} + D_{k,i} \quad (14)$$

Inequality (10) is obtained after some algebra form the QoS condition $R^t_{k,i} \leq \overline{R}_k$ and (9) and (14) as follows:

$$\begin{aligned} \overline{r}^t_{k,i} + \overline{d}^t_{k,i} + \sum_{\tau=1}^{t} \left(r^\tau_{k,i} + d^\tau_{k,i} \right) \leq \\ \left(\overline{R}_{k,i} - D_{k,i}\right) \left[C_i \mu_k \left(\overline{r}^t_{k,i} + \overline{d}^t_{k,i} + \sum_{\tau=1}^{t} \left(r^\tau_{k,i} + d^\tau_{k,i} \right) \right) - \tilde{x}^t_{k,i} \right] \Leftrightarrow \\ \sum_{\tau=1}^{t} (r^\tau_{k,i} + d^\tau_{k,i}) + \overline{r}^t_{k,i} + \overline{d}^t_{k,i} + \tilde{x}^t_{k,i} \left(\frac{\overline{R}_{k,i} - D_{k,i}}{1 - \left(\overline{R}_{k,i} - D_{k,i}\right) C_i \mu_k} \right) \geq 0 \end{aligned} \quad (15)$$

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

Finally, inequality (11) represents a constraint on the overall number of available reserved VMs, which 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).

Finally notice that, overall, problem (P_{st}) is a Mixed Integer Linear Problem (MILP), which can be efficiently solved by commercial solvers (an extensive scalability analysis is available in [25]).

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 Solution algorithm

In this section we describe the two algorithms used by our technique to solve the long- and short-term problems respectively.

4.1 Solution of the long-term problem

The long term problem solution is rather simple. As the optimization problem is clearly defined in the previous section, and due to the lack of relationship between the solutions of two consecutive hours, the algorithm can be simply defined as in Algorithm 1.

Algorithm 1 is supposed to run within one of the available data centers. It basically invokes a prediction function to forecast the incoming flow of requests for the next hour ($\tilde{\Lambda}_k$). The vector of real workloads needed for the forecasting is monitored locally to each data center on a hourly basis and is then sent to 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 of the DNS servers of each Cloud provider[3].

4.2 Solution of the short-term problem

The short-term problem is addressed using a controller implementing a receding horizon approach, outlined in Figure 4. This controller resides in every Cloud provider (data center) and operates independently from the other providers, unlike the solution for the long-term problem that 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 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. Notice that performance parameters are continuously updated at runtime in order

[3] https://www.nginx.com/resources/admin-guide/load-balancer/

to capture transient behavior, VMs network and I/O interference [21], and performance variability of the Cloud provider [22].

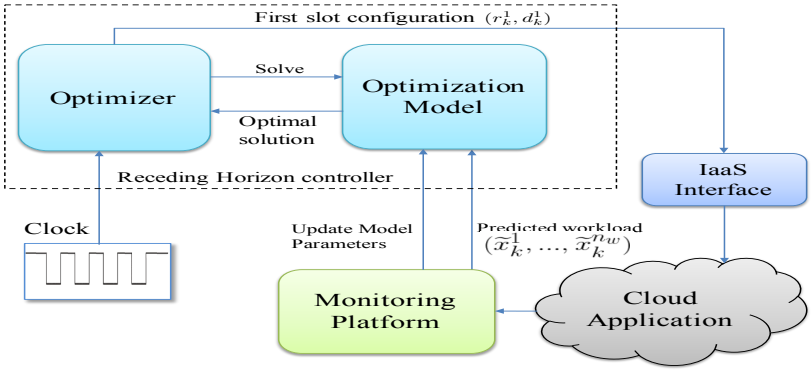

Figure 4: Receding horizon controller.

Algorithm 2 is a high-level description of the receding horizon approach we use to solve the short-term problem. The algorithm consists in four main steps, iteratively repeated to cover the overall time horizon. The first step (lines 2-8) initializes the main model parameters representing the state of the system and the predicted workload for the considered time window some model parameters. In particular, the system state is defined by the number of reserved and on-demand VMs available *free of charge* for each time slot of the observation window. It is worth to note that that, in order to manage the state of the system ($\overline{r}_{k,i}^t$, $\overline{d}_{k,i}^t$) in the execution of the algorithm for different time slots, we adopt the global parameters $N_{res,k,i}^t$, $N_{ond,k,i}^t$, which store the number of VM instances inherited from previous time slots and, therefore, available for free 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, $Scale(k, r_{k,i}^1, d_{k,i}^1)$. Finally (lines 12-15), the algorithm updates accordingly the system state, $N_{res,k,i}^{j+t}, N_{ond,k,i}^{j+t}$. 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.

5 Experimental Results

In this section our approach is compared with current state-of-art solutions implemented in modern Clouds and according to the auto-scaling policies often 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 analyses demonstrated that both the long-term and short-term algorithms scale almost linearly with respect to the number of sites and request classes. Systems up to 5 providers/data centers, 160 classes and considering 5 time slots can be solved in less

Algorithm 2 Receding Horizon Algorithm

1: **procedure** SOLUTION ALGORITHM
2: **for all** $k \in \mathcal{K}$ **do**
3: **for** $w \leftarrow 1, n_w$ **do**
4: $\tilde{x}^w_{k,i} \leftarrow$ ***GetPrediction*** (w, k)
5: $\overline{r}^w_{k,i} \leftarrow N^{t+w}_{res,k,i}$
6: $\overline{d}^w_{k,i} \leftarrow N^{t+w}_{ond,k,i}$
7: **end for**
8: **end for**

Initialization

9: ***Solve*** (P_{st}) $\leftarrow$ *Solving the current model*

10: **for all** $k \in \mathcal{K}$ **do**
11: ***Scale*** $(k, r^1_{k,i}, d^1_{k,i})$ $\leftarrow$ *Applying the changes according to the first time slot decisions*
12: **for** $j \leftarrow 1, n_c$ **do**
13: $N^{t+j}_{res,k,i} \leftarrow N^{t+j}_{res,k,i} + r^1_{k,i}$
14: $N^{t+j}_{ond,k,i} \leftarrow N^{t+j}_{ond,k,i} + d^1_{k,i}$
15: **end for**
16: **end for**

State update

17: **end procedure**

than 200 sec on a single core of an Intel Xeon Nehalem using CPLEX 12.2 solver. The complete scalability analysis is available in [25].

Section 5.1 presents the design of experiments. In Section 5.2 we describe the heuristics considered for comparison. Section 5.3 analyzes the results achieved in terms of the costs of the solutions and SLAs fulfillment considering analytical performance models. Section 5.4 provides an overview of the two algorithms performance tested using a simulator able to capture a more realistic scenario where performance may change due resource contention within a data center and Section 5.5 provides an experimental evaluation using a real system.

5.1 Design of experiments

The analyses performed in the following sections are intended to be representative of real Cloud environments. We used a large set of randomly generated instances, obtained varying the performance parameters in ranges used by other literature approaches [26], [6], [7] or observed in real applications as in [27, 28]. Such ranges are reported in Table 5.1. The bound on the number of reserved instances is set to 10 in order to lead the saturation of the available reserved resources and, then, considering the worst case solutions where also the on-demand resources are used (recall that by relaxing constraint (12) problems (P_{lt}) and (P_{st}) become 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 [2]. We decide to randomly generate the instance costs in order to replicate infrastructures

spread worldwide. Table 5.1 reports the ranges we adopted.

Regarding the application workload Λ_k, we used actual measurements coming from a large popular website that wants to remain anonymous for privacy reasons. Workload can be characterized by 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 [29], [27], [30] and [31], we produced a different workload for each WS application k, by scaling the peak value of each request class. We further added some noise as in [29], thus extending the original trace set with the aim of analyzing the behavior of WS applications under different workload conditions.

The workload prediction $\widetilde{\Lambda}_k$ is obtained by adding white noise to each sample Λ_k, as in [27], [32] (the noise is proportional to the workload intensity Λ_k). For what concerns the short time scale, the predictions $\tilde{x}^t_{k,i}$ in a real context become 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 $t \in [1 \dots n_w]$ (further details are reported in the next section). Our choice of applying white noise is consistent with [31] and provides the benefit of being independent from the choice of the prediction technique.

5.2 Heuristics

In the following sections we perform a cost-benefit evaluation of our algorithms with respect to other heuristics proposed in the literature and based on utilization thresholds.

Heuristic 1: it derives from [6], [7] and is currently implemented by some IaaS providers (see, e.g., Amazon AWS Elastic Beanstalk [8]). The heuristic implements an algorithm for auto-scaling the number of instances in order to handle the workload variation. As in our approach, capacity allocation is performed over the overall time horizon considering an observation window $\mathcal{T}_w$ 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 [$\overline{U}_1$,$\overline{U}_2$].

$$\overline{U}_1 \leq U^t_{H1} \leq \overline{U}_2 \qquad \forall t \in [1..n] \tag{16}$$

As in [27], we considered multiple values for $[\overline{U}_1, \overline{U}_2]$, as it will be discussed more in details later.

Heuristic 2: it is based on a more accurate evaluation of the utilization thresholds. Instead of considering fixed values for $\overline{U}_1$ and $\overline{U}_2$, they are derived from the response time threshold $\overline{R}_k$. In particular, by considering the response time formula (14) and the constraints on the average response time (11) we obtain the following condition on the VMs utilization:

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

In other words, $\overline{U}_k$ is the VMs utilization which corresponds to the average response time $\overline{R}_k$ at provider i. So, if we denote with α and β, respectively, the coefficient of lower and upper bound thresholds, with $\alpha < \beta$, we define the utilization thresholds as:

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

In this way, we obtain different thresholds for different WS classes. In the experiments we adopt different thresholds characterized by different values of α and β, as we will described in the following section.

5.3 Cost-Benefit Analysis

We perform a cost-benefit evaluation of our approach considering other heuristics with a twofold aim: on one hand we compared the cost of our solution against other state-of-the-art techniques. On the other hand, we assessed the impact of the number of time slots within the sliding window on the CA solution.

As performance indicators we used the overall virtual machines cost and the number of SLA violations. The test case considers a single IaaS provider (and hence relies on Algorithm 2 only) and it is based on a 24 hours time span and each hour is divided into time slots of 10 or 5 minutes, according to the time scale under analysis. The incoming traffic models used in the case under study are grouped into two categories: "normal traffic model" and "spiky traffic model," corresponding to two different traces obtained from real logs. In both cases, the traces, presented in [27], are characterized by the presence of daily peaks around 10.00am and 2.00pm but in the normal traffic model the number of client requests exhibits a slow increase or decrease rate. Vice versa, the spiky workload is characterized by larger variations between consecutive samples. Furthermore, for each workload trace we considered both a low 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 $t \in [1 \dots n_w]$. Furthermore, the level of noise of the workload sample $\tilde{x}_k^t$ is proportional to the workload Λ_k and increases with t as reported in Table 5.3. A noise level up to 45% has been considered, 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.

The alternative solutions considered in our comparisons are the following:

- Our short-term algorithm (*S-t Algorithm*): the time scale T_{slot} has been varied between 5 and 10 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. At a first though this could seem the top performing approach, but it is not necessarily cheaper than other solutions. Indeed, if the prediction slightly underestimates the real traffic, our algorithm or other heuristics could

result in a cheaper solution than the one determined by the Oracle, but, of course, they would provide a larger number of SLA violations.

- Heuristic 1 (*Heu1*): the heuristic adopts the same time horizon, time step, and VM life span (of one hour for each instantiated VM) as in our solution. The number of instances is determined such that the utilization of each running instance is between some given thresholds. The thresholds used in the experiment are: $(\overline{U}_1, \overline{U}_2) = (40\%, 50\%)$, $(50\%, 60\%)$, and $(60\%, 80\%)$ as considered in [7, 27].

- Heuristic 2 (*Heu2*): this heuristic differs from the previous one only for the way the thresholds are calculated. The thresholds coefficient we considered are: $(\alpha, \beta) = (0.9, 1.2)$ and $(0.8, 1.3)$.

In the following, we first focus our attention on the costs achieved by the considered alternatives and then on the SLA violations. The reference scenario under analysis is composed of 10 different classes.

In the following figures we present the mean solution cost evaluated over multiple test executions for a 24 hours time horizon.

Precisely, we executed three different tests for each configuration of time scale, workload type, noise and thresholds, for a total number of 72 runs; each run calculated the cost considering the overall time horizon. For space limitations we report here detailed results only for the spiky workload case. The complete analysis is available in [25]. Figures 5 and 6 show the results obtained with the adoption of a time scale of 5 minutes, instead Figures 7 and 8 display the values we obtained by using a 10 minutes time scale. Being the level of noise proportional to the incoming workload Λ_k, in Figures 6, 7 and 8 we did not consider a time window of 5 step ahead, since, in the case of spiky workload, this would make the prediction to become unreliable. Notice that our approach is competitive with the Oracle and we record the best performance of Algorithm 2 with a time scale of 5 minutes. The cost difference between our solution and the Oracle, which is more relevant in case of spiky traffic (Figures 5, 6), denotes how strongly the traffic prediction is affected by the noise with the increase of observation windows size. 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.

Figures 7 and 8 describes the evolution of the average solution cost with a time scale of 10 minutes. The performance trend that we noticed in the previous figures is confirmed even if we have to underline that the difference between our Short-term algorithm and Heuristic 2 is significantly reduced.

Finally, it is worth to be noticed that in a normal workload conditions the 10 minutes time scale is the best suited solution; in case of a more spiky traffic, it is more beneficial to employ a time scale of 5 minutes.

In our analysis, we considered also the number of SLA violations during the 24 hours time horizon (i.e., the number of times the average response time during a time slot was above the threshold). This analysis allows to understand whether the receding horizon technique leads to a reduction of violations with respect to single step optimization (i.e., obtained with $n_w = 1$), which characterizes most of other literature

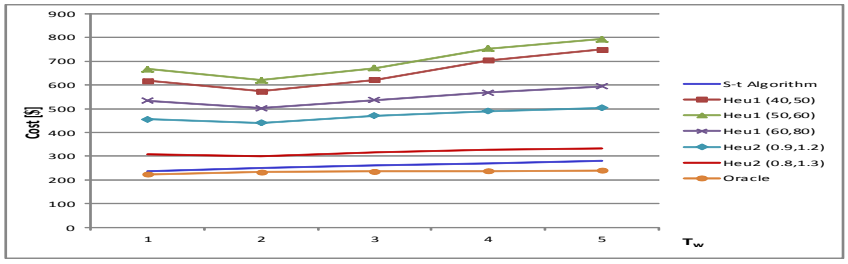

Figure 5: Solution cost, $T_{slot} = 5min$, spiky traffic and low noise level.

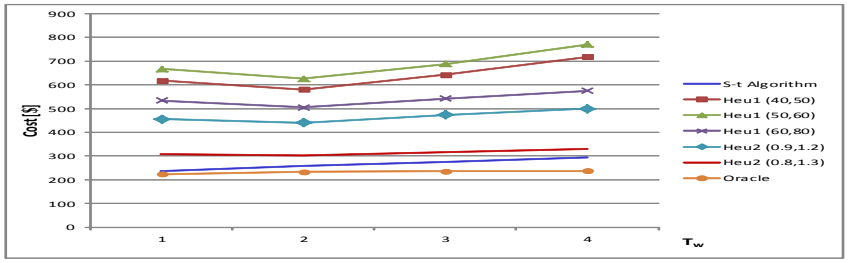

Figure 6: Solution cost, $T_{slot} = 5min$, spiky traffic and high noise level.

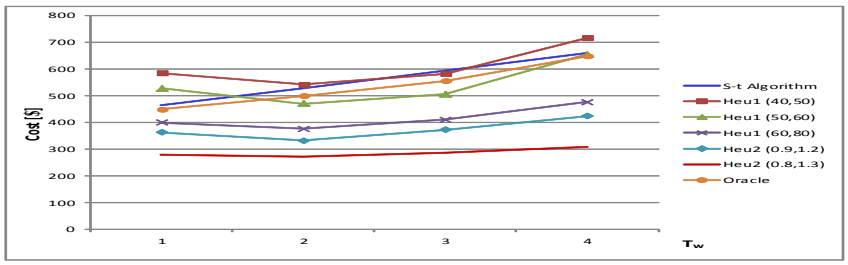

Figure 7: Solution cost, $T_{slot} = 10min$, spiky traffic and low noise level.

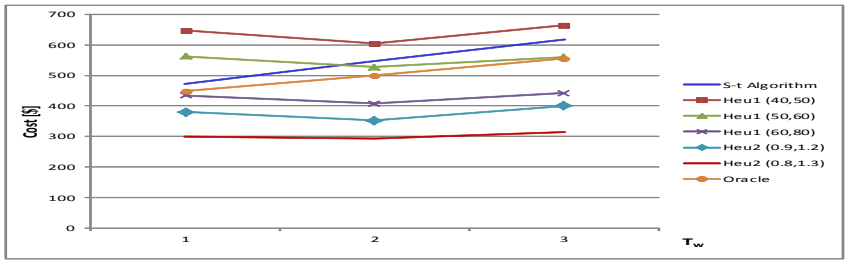

Figure 8: Solution cost, $T_{slot} = 10min$, spiky traffic and high noise level.

approaches. Results are reported in Tables 7, 7, 7 and 7. Increasing the observation window size n_w, the percentage of SLA violations for our solution decreases. Furthermore, our solution exhibits very small percentages of violations with respect to Heuristic 2, both (0.9, 1.2) and (0.8, 1.3) cases, even in the presence of spiky and high

noisy workloads, where, Heuristic 2 shows better results in terms of resource costs. Therefore, the receding horizon technique allows improving the performance of the system in terms of SLA violations. Heuristic 1, instead, is very conservative: it is more expensive than our approach, but it satisfies the QoS constraints almost in any condition of traffic and noise.

Moreover, considering the comparison between the average results obtained by the two time scale under analysis, the 5 minutes time scale guarantees better performance in terms of SLA violations: 8 minutes (0.556%) over 24 hours for the normal traffic and 15.3 minutes (1.065%) for the spiky one. Furthermore, using two or more forward steps the performance improves: A maximum of 2.5 minutes (0.174%) for the normal traffic and 7.3 minutes (0.509%) for the spiky one. This means that in terms of SLA violations our algorithm performs significantly better with the adoption of at least two forward steps. Finally a more in-dept analysis demonstrates that our algorithm provides the best results by making use of an observation windows with three steps.

5.4 System simulation

While the analytical evaluation of the proposed algorithm (i.e., based on M/G/1 formula) reported in the previous section provides a first confirmation of the viability of our proposal, an analytical-only validation is not capable of taking into account the effect of the interaction among VMs that occur in a real data center. To evaluate our proposal in a more realistic scenario and considering multiple providers/data centers, we use a simulator specifically designed to capture the variable performance of real data centers. Specifically, we first describe the simulator and its setup and then we provide an in-depth discussion of the results obtained through simulation.

5.4.1 Simulation Setup

The proposed simulator implements the data center model described in Section 2, and illustrated in Figure 2. In particular, we consider a reference scenario with three heterogeneous data centers, 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 long-term algorithm partitions the incoming workload according to the processing capacity of each data center while the VMs management in each data center is handled by the short-term algorithm.

To validate our proposal we rely on a discrete event simulator based on the Omnet++ framework [33] that has been developed ad-hoc for this purpose. In particular, we introduce in our simulator a support to capture the performance degradation that appears randomly in Cloud data centers due resource contention and imperfect performance isolation between VMs. To this aim our simulator includes Random Environments (REs) [9]. 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 [9, 34] 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 [35].

Within the simulator we implement REs 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 $D_k^{fast} < D_k^{slow}$. We consider a transition probability between the two stages: p^{fast} is the transition probability from the state *slow* to state *fast*, while p^{slow} is the probability of the opposite transition.

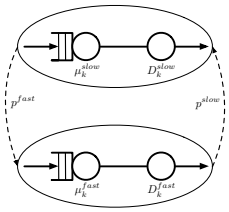

Figure 9: Random Environment modeling Cloud resource contention.

In order to model these effects in our simulator, we collected data from several experiments on applications running on an Amazon EC2 infrastructure. To this aim we executed a computationally-intensive application on VMs of different sizes and we monitored the response time of the application as well as the system parameters. In particular, we consider CPU utilization and CPU steal (this latter is the fraction of VM CPU time that is claimed by the hypervisor). Our experiments found that when the application runs on large VM instances there is no performance degradation and no CPU stealing form 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 shown in Figure 11, we observe under constant incoming workload that, after a period where with full processing power, the hypervisor intervenes capping the CPU consumption of the VM, as shows by the performance degradation and by the increase of the CPU steal parameter. For an application that is not completely CPU-bound, such as a web-based application where the load is distributed by a load balancer over 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 degrade performance that is easily modeled by the REs model of our simulator.

The experiments on the real system are used to model the parameter of the REs in the simulator. The resulting values are reported in Table 10. In particular we use the performance degradation ratio to quantify the impact of congestion on the VM performance (both in terms of processing time and of delay) and we model the transitions between $slow$ and $fast$ states 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 5.1.

An additional element that we capture in our simulator is the process of client

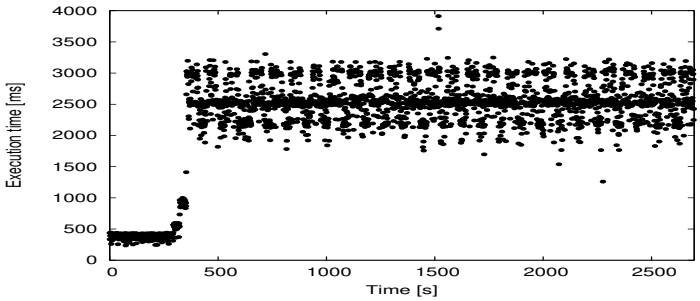

Figure 10: Execution time under CPU steal effect in Amazon EC2 medium instance

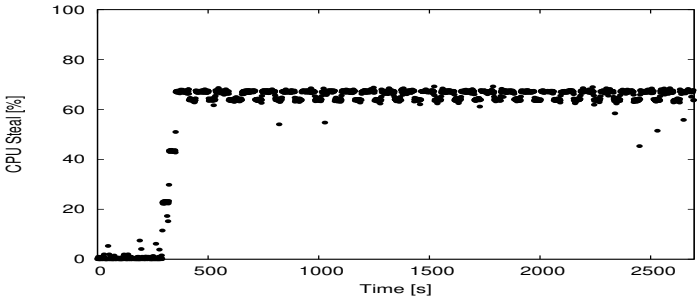

Figure 11: CPU steal in Amazon EC2 medium instance.

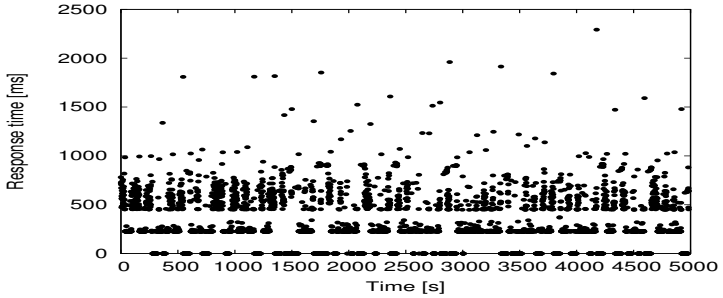

Figure 12: Performance degradation in a multi VM web server application.

requests dropped from the waiting queue. We consider that client requests may leave the system without being serviced for the following two reasons: (1) because a timeout To is expired and 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 effect is typically configured in web servers such as Apache httpd[4] to avoid thrashing conditions. In our experiments, we explore several scenarios for different combinations of To and Q, as detailed in the following subsection. Table 10 provides a summary of the parameters used in our simulations.

[4]http://httpd.apache.org

5.4.2 Sensitivity to queue length and timeout

We now evaluate through our simulator 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, we consider two scenarios for the timeout parameter, that is, a *short timeout* scenario where the timeout is five time 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 we consider multiple queue lengths Q and observation window sizes $\mathcal{T}_w$ (the observation window size is considered in the receding horizon algorithm used to solve the short-term problem).

Table 11 reports the result of these experiments for the two considered scenarios. If we consider the short timeout scenario we can observe three interesting results. First, for every considered combination of parameters we have no SLA violations. If we focus on the percentage of dropped requests, we observe that the values range from 0.3% to 1.8%, which outperforms results that can be achieved in other literature proposals [36].

The second interesting finding is that, the percentage of dropped requests decreases as the queue length grows. This result is easily explained by considering that longer queues means that the system is able to better manage incoming peaks of requests without rejecting the incoming load. Finally, we observe that the number of dropped requests is reduced also as a function of the observation window used in 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.

The long timeout scenario basically confirms that main findings of the experiments with a short timeout. However, comparing the results of the two timeout scenarios, we observe that the percentage of dropped requests for the long timeout scenario ranges from 4.2% to 0.7%, which is significantly higher if compared with 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 in a short time and have a smaller opportunity to consume resources.

Furthermore, we observe that, when the queue length grows beyond 30 requests, besides the reduction in the number of dropped requests, we may experience a small amount of SLA violations (below 0.1%). This result can be explained by considering that the long queue length determines an increase in the requests within the system, resulting in a growth in the response time that determines the SLA violations.

5.5 Prototype Environment Analysis

The experimental analysis presented throughout this section has been performed considering the *Modelio Constellation*[5] modeling platform developed by Softeam within the MODAClouds project. Modelio Constellation is a web-based software-as-a-service

[5] https://link.springer.com/chapter/10.1007/978-3-319-46031-4_12/fulltext.html

application modeling environment, which includes four main components. The *Administration Server* exposes a web service interface to provide the users with a GUI to retrieve, modify and update the available projects and read their configuration. This component uses the *Administration Database* 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 due to the fact that the 80% of the requests have been found to be SVN reads, we decided to deploy just the *HTTPAgent* in a dedicated VM and to consider this simplified scenario for the evaluation of our short term algorithm. Along this path, we measured the resource demand of the *HTTPAgent*, that we found to be around 40 ms (in particular $D_k = 5$ ms, $\frac{1}{\mu_k} = 35$ ms).

We considered two scenarios. We first injected a ramp-like workload (Figure 5.5) with a peak around 30 requests/s and then we evaluated the system with the workload from real users (Figure 13) obtained from the logs of a pre-production environment, that is basically a bi-modal workload with 100 requests/s at its peaks, located in the central part of the day. In this second scenario, we shrunk to 4 hours the original 24 hour lasting workload 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 our short term algorithm to react to workload fluctuations considering the SOFTEAM *HTTPAgent* component deployed on Amazon m3.large VMs. We used Apache JMeter to inject the two previously described workloads. 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 min. 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 [37], acting at 10s time scale.

The short term algorithm started up to 11 VM instances. The percentage violations of the average response time measured at runtime were 1.9%, if the average is evaluated at 10 s time scale and 0% if the average is evaluated at 5 min time scale. 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.

6 Related Work

Several approaches have been developed 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 [38] 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 [39] 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 [40] 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. A multi-Cloud perspective is considered in [41], where an on-line approach is used to address problem of allocating VMs over a distributed infrastructure. The work in [42] 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 slightly different approach, extending the problem to multiple Cloud data centers and focusing on scientific work-flows is presented in [43]. However, most of these mechanisms do not follow the split long-term and short-term problems to be solved separately to provide a scalable and accurate resource provisioning. In [31] the VM placement problem for a PaaS is solved at multiple time scales through a hierarchical optimization framework similar to the present paper. However, our work aims at providing a different vision to the problem, more tailored to the multi-Cloud scenario.

A methodology to integrate workload burstiness in performance models is proposed in [44]. The workload model share some common traits with [45], even if the focus in this latter paper is more oriented towards a container-based virtualization scenario. 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 [46] 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 [47] 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 [48]. 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 the adoption of two time scales and of receding horizon solutions. The work in [49] 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.

7 Conclusions

In this work we proposed a capacity allocation technique able to minimize the execution cost of Cloud applications guaranteeing the respect of the SLA. An extensive analysis that takes into account multiple factors as different workloads and system configurations has been provided demonstrating that our approach 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 [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 we can achieve a maximum of 8 minutes average response time violation over 24 hours (versus up to 260 minutes of other approaches). 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 condition.

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. From 2012 he is Postdoctoral Fellow at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano. His research work focus on modeling and optimization of complex real-life problems mainly arising in the fields scheduling and planning, and more recently resource management for Cloud based and data intensive systems under constraints of quality of service.

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] 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 Distr. Computing*, vol. 72, no. 6, pp. 796 – 808, 2012.

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

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

[7] 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*, vol. 12, no. 1, pp. 45–57, 2009.

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

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

[10] 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*.

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

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

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

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

[15] Amazon Inc., "Elastic Load Balancing," http://aws.amazon.com/elasticloadbalancing/.

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

[17] 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.

[18] 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.

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

[20] 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.

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

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

[23] 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.

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

[25] 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.

[26] 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*, vol. 70, no. 4, pp. 344 – 362, 2010.

[27] 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*, vol. 72, no. 6, pp. 796 – 808, 2012.

[28] 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.

[29] 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.

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

[31] 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*, vol. 10, no. 5, pp. 253–272, 2013.

[32] 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*, vol. 38, no. 5, pp. 1221 –1233, oct. 2008.

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

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

[35] 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.

[36] Y. Amannejad, D. Krishnamurthy, and B. H. Far, "Predicting Web service response time percentiles," in *CNSM 2016 Proceedings*.

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

[38] 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*.

[39] H. Goudarzi and M. Pedram, "Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems," in *IEEE CLOUD 2011 Proceedings*.

[40] 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*.

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

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

[43] 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 Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 647–661, March 2017.

[44] 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*, vol. 38, no. 5, pp. 1040 –1053, sept.-oct. 2012.

[45] O. Adam, Y. C. Lee, and A. Zomaya, "Stochastic resource provisioning for containerized multi-tier web services in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. Pre-print, pp. 1–1, 2017.

[46] M. Ferber, T. Rauber, M. Torres, and T. Holvoet, "Resource allocation for cloud-assisted mobile applications," in *IEEE CLOUD 2012 Proceedings*.

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

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

[49] 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 *USENIX*, 2011.

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 sec) for processing WS class k requests at provider i
$R_{k,i}$	Average response time (measured in sec) for WS class k request at provider i
$\overline{R}_k$	Average response time threshold (measured in sec) for WS class k request
W_i	Maximum number of *reserved* instances available, at provider i
μ_k	Maximum service rate (measured in requests/sec) 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/sec) for WS class k, at provider i and at time slot j
$\tilde{x}^t_{k,i}$	Local arrival rate prediction (measured in requests/sec) 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 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

Table 3: Performance parameters

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

Table 4: Cost parameters

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

Table 5: Noise levels adopted.

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

Table 6: Response Time percentage violations $T_{slot} = 5$ min, spiky traffic and high noise.

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 for $T_{slot} = 5$ min, spiky traffic and low 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 $T_{slot} = 10$ min, spiky traffic and low 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 9: Response Time percentage violations for $T_{slot} = 10$ min, spiky traffic and high 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 10: 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	$5, 10 \times \frac{1}{\mu^{fast}}$

Table 11: Sensitivity to queue length and timeout

	$\mathcal{T}_w = 1$		$\mathcal{T}_w = 2$		$\mathcal{T}_w = 3$		$\mathcal{T}_w = 4$		$\mathcal{T}_w = 5$	
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

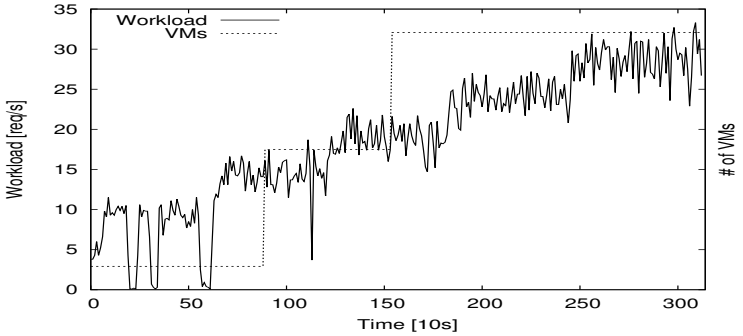

Figure 13: VMs allocation, 5 minutes time scale, ramp workload and low noise level.

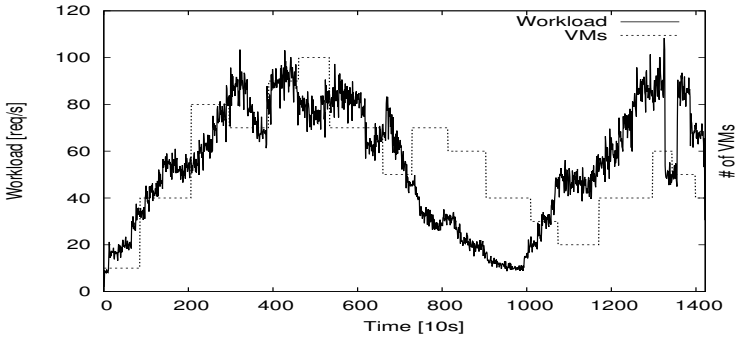

Figure 14: VMs allocation, 5 minutes time scale, real bimodal workload and low noise level.

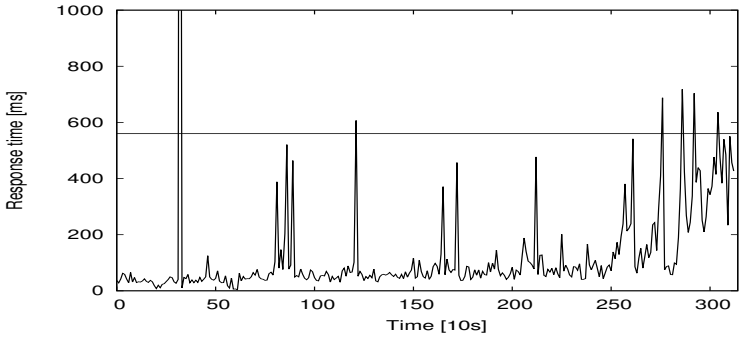

Figure 15: Response times, 5 minutes time scale, ramp workload and low noise level.

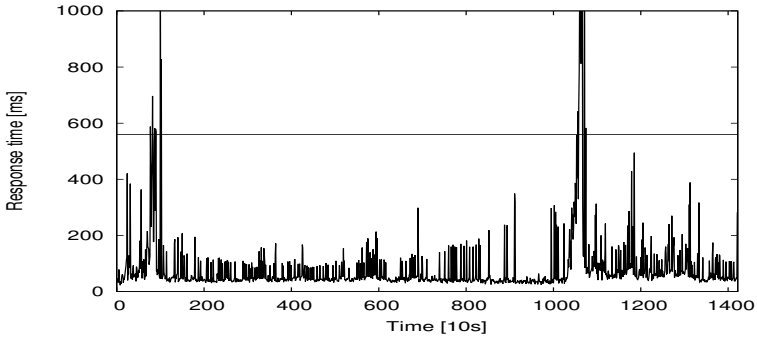

Figure 16: Response times, 5 minutes time scale, real bimodal workload and low noise level.