Multi-time Scale Distributed Capacity Allocation and Load Redirect Algorithms for Cloud Systems

Danilo Ardagna*, Sara Casolari**, Michele Colajanni**,Barbara Panicucci*
*Politecnico di Milano, Dipartimento di Elettronica Informazione
**Università di Modena e Reggio Emilia, Dipartimento di Ingegneria dell'Informazione
Email: {ardagna,panicucci}@elet.polimi.it, {sara.casolari,michele.colajanni}@unimore.it

Abstract

Resource management remains one of the main issue in cloud computing because system resources have to be continuously allocated to handle workload fluctuations while guaranteeing Service Level Agreements (SLA) to the end users. In this paper, we propose capacity allocation algorithms able to coordinate multiple distributed resource controllers operating in geographically distributed cloud sites. Capacity allocation solutions are integrated with a load redirection mechanism which, when necessary, distributes incoming requests among different sites. The overall goal is to minimize the costs of allocated resources in terms of virtual machines, while guaranteeing SLA constraints expressed as a threshold on the average response time. We propose a distributed solution which integrates workload prediction and distributed non-linear optimization techniques. Experimental results show how the proposed solutions improve other heuristics proposed in literature without penalizing SLAs, and they are close to the global optimum which can be obtained by an oracle with a perfect knowledge about the future offered load.

Keywords: Cloud systems, Performance modeling, Resource management, Capacity allocation, Load balancing, SLA.

I. Introduction

Cloud computing is a paradigm that aims at streamlining the on-demand provisioning of software, hardware, and data as services, and providing end-users with flexible and scalable services accessible through the Internet [28].

Cloud infrastructures live in an open world characterized by continuous changes in the environment and requirements they have to meet. Continuous changes occur autonomously and unpredictably, and they are out of control of the cloud provider. Nevertheless, cloud-based services must be provided with different Service Level Agreements in terms of reliability, security and performance. In this paper, we focus on solutions for performance guarantee that are able to dynamically adapt the resources of the cloud infrastructure in order to satisfy SLAs and to minimize costs. To this purpose, we integrate workload prediction models into capacity allocation techniques that are able to coordinate multiple distributed resource controllers working in geographically distributed cloud sites. These allocation techniques can work together with a dynamic load redirection mechanism which, during peak loads, moves requests from heavily loaded sites to other sites. The proposed request distribution algorithms aim at optimizing the average response time of user requests and satisfy SLA requirements.

In cloud architectures that are characterized by geographically distributed systems, any centralized approach for capacity allocation and load balancing is subject to critical design limitations including lack of scalability and expensive communication costs [20]. Therefore, distributed solutions are mandatory [7], [34], [57], [31]. This paper adopts a distributed approach where, capacity allocation and load redirect are modelled as non-linear programming problems. The optimization problems are solved by implementing decomposition techniques which are integrated with traffic predictive models, used to estimate the incoming workload at each site and the requests rate redirected from heavily loaded sites to the others.

We compare our approach with other heuristics proposed in the literature [23], [59], [58]. A large set of experimental results demonstrates that the proposed solutions save costs and do not incur in SLA violations. It is also worth to observe that our solutions are close to the global optimum that can be obtained by an oracle having a perfect knowledge about the future workload.

The remainder of the paper is organized as follows. Section II formalizes the problem. Section III describes our reference framework and design assumptions. The formulation of the optimization problem is presented in Section IV. The prediction techniques are introduced in Section V. The experimental results demonstrating the improvements of the proposed solutions are reported in Section VI. Other literature approaches are discussed in Section VII. Conclusions are finally drawn in Section VIII.

II. Problem statement

In this paper we take the perspective of a provider which offers transactional Web-based services (WS) hosted by multiple sites of an Infrastructure-as-a-Service (IaaS) provider. The hosted services represent heterogeneous applications in terms of resource demands, workload intensities, and SLA requirements.

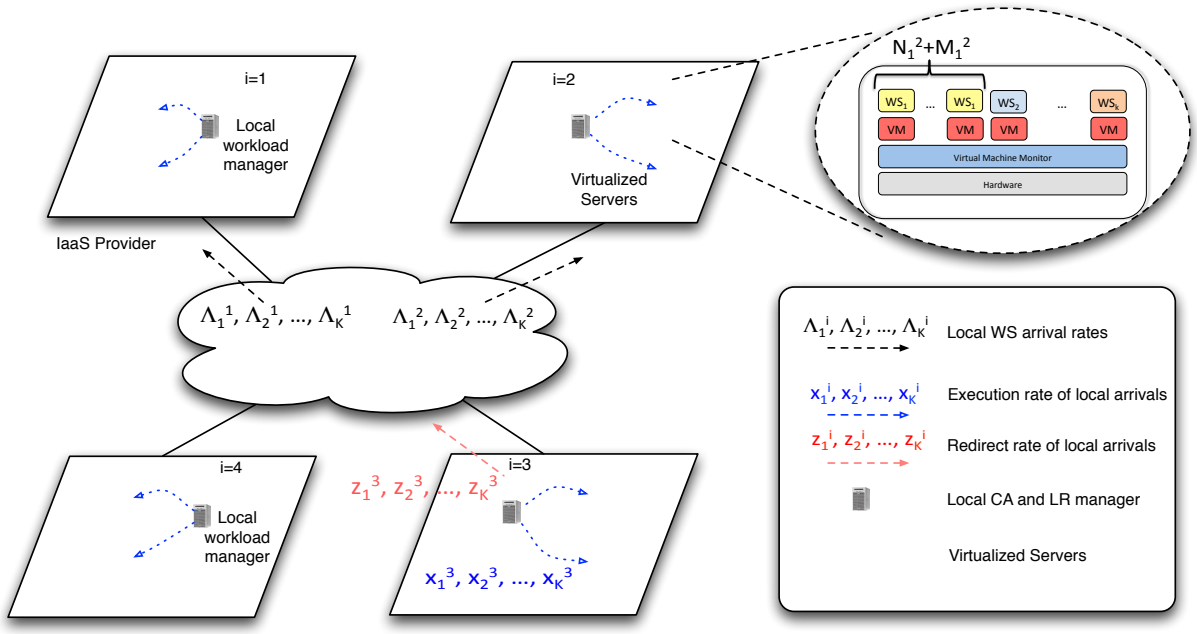

Fig. 1. Cloud System Reference Framework.

Services with different SLAs and workload profiles are categorized into independent classes.

We assume that an SLA contract associated with each WS class k is established between the WS provider and its end users. This contract specifies the SLA levels expressed in terms of average response time R_k that the WS provider must meet while responding to end users requests for a given service class. Overall, the system serves a set K of WS classes and average response time thresholds that have to be guaranteed on a five minutes time scale are denoted by $\overline{R}_k$.

Applications are hosted on virtual machines (VM) which are provided on a pay-per-use basis by the IaaS provider. For the sake of simplicity, we assume that each VM hosts one Web service application. Multiple VMs implementing the same WS class can run in parallel at each physical location. We assume that the VMs are homogeneous in terms of RAM and CPU capacity [4] and evenly share the incoming workload (this corresponds to the solution currently implemented by IaaS providers [6]). Furthermore, services can be located on multiple geographically distributed sites (see Figure 1). For example, Amazon Elastic Compute Cloud (EC2) allows software providers to deploy VMs on five world regions.

IaaS providers usually charge software providers on a hourly basis [4]. Hence, the WS provider has to face the Capacity Allocation (CA) problem which consists on determining every hour the optimal number of VMs for each WS class in each IaaS site according to the average load predicted on a

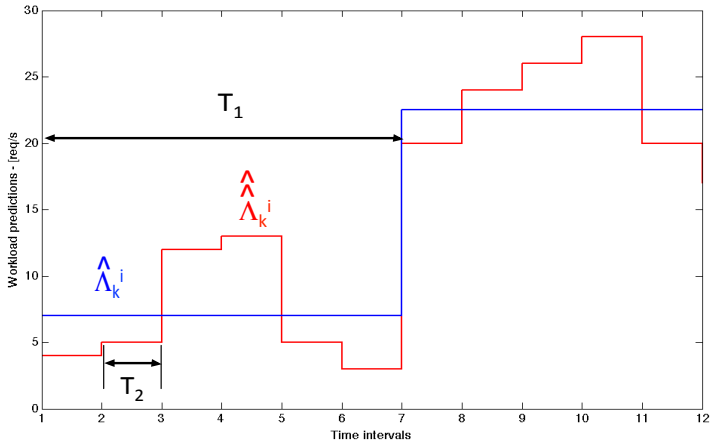

Fig. 2. Cloud System Reference Framework.

hourly basis, while guaranteeing SLA constraints. We denote by T_1 the mid-long time scale adopted for VM provisioning. If the resources of a site are insufficient (e.g., because of an unpredictable workload peak), incoming requests can be even redirected to other sites. As in other approaches, dynamic Load Redirection (LR) [12], [59] is performed periodically every $T_2 << T_1$ time instants (see Figure 2) at a more fine-grained time scale (e.g., 5 to 10 minutes) on the basis of a short-term prediction of future WS workloads [1], [16] or can be triggered by a monitoring system in order to react to unexpected events, such as system failures.

By considering two different time scales, we are able to capture two types of information related to the IaaS sites [59]. The fine grain workload traces exhibit a high variability due to the short-term variations of the typical Web-based workload and for this reason, the fine-grained time scale provides useful information for the dynamic load redirections. 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 allow us to use the mid-long time scale algorithms to predict the workload trend that represents useful information for the capacity allocation algorithm.

We denote by I the set of IaaS sites. For simplicity, we assume that at each site VMs are homogeneous in terms of computing and storage capacity. Even in the case of heterogeneous VMs, cloud providers have a limited set of available configurations, say S. Hence, a site with heterogeneous resources can be modelled as S sites with homogeneous resources. The capacity of VMs at site i is denoted by C^i, while for each site i and WS class k we denote with $\widehat{\Lambda}_k^i$ the arrival rate predicted at the time scale T_1 coming

from the time zone where the site is located, while we denote through $\widehat{\widehat{\Lambda}}_k^i$ the corresponding prediction at the time scale T_2 (see Figure 2). Finally, Λ_k^i will indicate the real local arrival rate.

The objective of the CA problem is to determine the number of VMs able to serve $\widehat{\Lambda}_k^i$ requests/sec, while minimizing VMs costs and guaranteeing that $R_k \leq \overline{R}_k$. We assume that a WS provider can establish two different contracts with the IaaS provider. Namely, it may be possible to access VMs on a pure *on-demand* basis and the WS provider will be charged on a hourly basis (see e.g., Amazon EC2 *on-demand* pricing scheme, [4]). Otherwise, it may be possible to pay a fixed annual *flat* rate for each VM and then access the VMs on a pay-per-use basis with a fee lower than the pure *on-demand* case (see e.g., Amazon EC2 reserved instances pricing scheme, [4]). The time unit cost (e.g., $ per hour of VM usage) for the use of *flat* VMs at site i is denoted by $\overline{c}^i$, while the cost for VMs *on demand* will be denoted by $\widetilde{c}^i$, with $\overline{c}^i < \widetilde{c}^i$. The CA problem solution determines every T_1 time unit the number of *flat* VMs to be allocated to WS class k at site i, N_k^i, and the number of *on demand* VMs to be allocated to class k at site i, M_k^i. We will denote with $\overline{N}^i$, the number of *flat* VMs available at site i obtained through the annual flat contract.

On the other hand, the LR problem aims at determining (every T_2 time instants) the execution rate of local arrivals for class k at site i, x_k^i, and the redirect rate of class k at site i toward the other sites, z_k^i, in order to satisfy the prediction $\widehat{\widehat{\Lambda}}_k^i$ for the local arrivals, while guaranteeing that $R_k \leq \overline{R}_k$.

For the sake of clarity, the notation adopted in this paper is summarized in Table I.

III. Reference framework and design assumptions

Our dynamic CA and LR techniques combine a workload predictor and an optimization model.

In the following we model each WS class hosted in a VM as an M/G/1 queue in tandem with a delay center [18], as done in [40] and we adopt the typical assumption in Web service containers [44], [3], that requests are served according to the processor sharing scheduling discipline. Multiple VMs can run in parallel to support the same application. In that case, the workload is evenly shared among multiple instances (see Figure 3). As discussed in [40], the delay center D_k allows us to model network delays and/or protocol delays introduced in establishing connections, etc.

Future performance for each WS class are obtained on the basis of the prediction of forecasted workloads. The optimization model uses these estimates to determine the number of VM instances N_k^i and M_k^i, the execution rate of local arrivals for each class x_k^i and, possibly, the workload redirected to other sites z_k^i.

System parameters

I	Set of sites
K	Set of WS classes
C^i	VM instances capacity at site i
$\overline{c}^i$	Time unit cost for *flat* VMs at site i
$\widetilde{c}^i$	Time unit cost for *on demand* VMs at site i
$\overline{N}^i$	Number of *flat* VMs available at site i
T_1	Long term CA time horizon
T_2	Short term LR time horizon
Λ_k^i	Real local arrival rate for WS class k at site i
$\widehat{\Lambda}_k^i$	Local arrival rate prediction for WS class k at site i at time scale T_1
$\widehat{\widehat{\Lambda}}_k^i$	Local arrival rate prediction for WS class k at site i at time scale T_2
$\widehat{\widehat{\lambda}}_k^i$	Estimation of the overall request redirect to site i for WS class k at time scale T_2
μ_k	Maximum service rate of a capacity 1 VM for executing WS class k requests
D_k	Queueing delay for processing class k requests
$d_k^{i,j}, i \neq j$	Network transfer time for redirecting class k requests from site i to site j
$g_k^{i,j} = \frac{1}{d^{i,j}}, i \neq j$	"Conductance" of the communication link (i,j) for class k requests
$G_k^i = \sum_j g^{i,j}, i \neq j$	"Equivalent conductance" seen from site i to the other sites for class k requests
R_k^i	Average response time for executing WS class k request at site i
$\overline{R}_k$	WS class request k average response time threshold

Decision Variables

N_k^i	Number of *flat* VMs allocated for class k request at site i
M_k^i	Number of *on demand* VMs allocated for class k request at site i
x_k^i	Execution rate of local arrivals for WS class k request at site i
z_k^i	Redirect of WS class k request at site i toward other sites

TABLE I

PARAMETERS AND DECISION VARIABLES FOR CAPACITY ALLOCATION AND LOAD REDIRECT PROBLEMS.

For the sake of simplicity, if the workload is redirected to other sites, the fraction of workload to individual sites is inversely proportional to the network transfer time for redirecting class k requests from site i to site j, $d_k^{i,j}$ or equivalently is directly proportional to the "conductance" $g_k^{i,j}$ of the class k request network link between site i and j defined as $g_k^{i,j} = 1/d_k^{i,j}$. In other words, if we define the "equivalent conductance" G_k^i at site i for class k as:

$$G_k^i = \sum_{j \in I, i \neq j} g_k^{i,j}$$

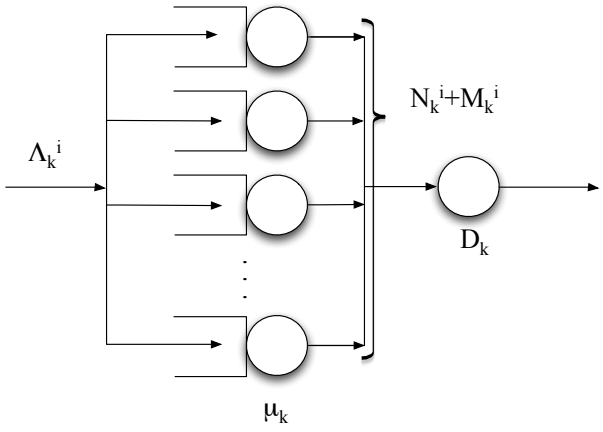

Fig. 3. System Performance Model.

the overall load at site i due to the redirect of other sites is given by:

$$\sum_{j \in I, j \neq i} \frac{g_k^{j,i} z_k^j}{G_k^j}.$$

At the time scale T_2, the total rate of class k requests executed at site i is the sum of the requests executed from local arrival, i.e. $x_k^i \leq \Lambda_k^i$, and the requests executed from the redirect which, according to the previous equation is given by:

$$x_k^i + \sum_{j \neq i} \frac{g_k^{j,i} z_k^j}{G_k^j}. \tag{1}$$

In other words, in our LR scheme requests can be redirected only once. Otherwise multiple hops could penalize some individual requests, thus increasing the overall response time variance of requests within the same WS class.

The next section formulates the optimization problems and devises a decentralized solution for the CA and LR problems. Section V discusses the prediction models adopted in this paper.

IV. Optimization Problem Formulation

As discussed in Section II, Capacity Allocation and Load Redirect are performed with different time scales. We formulate the Capacity Allocation problem in the next Section, while the Load Redirect approach is presented in Section IV-B.

A. Capacity Allocation problem

The CA problem is solved with T_1 time period. It aims at minimizing the overall costs for flat and on demand VM instances of multiple distributed IaaS sites, while guaranteeing that the average response

time of each class is lower than the SLA threshold. The CA determines the number of VMs N_k^i and M_k^i required to serve the arrival rate $\widehat{\Lambda}_k^i$. In this phase the traffic redirected to other sites is not considered. Indeed, preliminary results have shown that the LR mechanism, even if significant at the lower time scale T_2, introduces a limited increment to each class local incoming workload $\widehat{\Lambda}_k^i$ at time scale T_1.

If μ_k denotes the maximum service rate of a capacity 1 VM for executing WS class k requests, the response time for executing locally the WS class k at site i is given by $R_k^i = \frac{1}{C^i\,\mu_k - \frac{\widehat{\Lambda}_k^i}{N_k^i + M_k^i}}$. In particular it must be (M/G/1 equilibrium condition) $\widehat{\Lambda}_k^i < C^i\,\mu_k\,(N_k^i + M_k^i)$, and the total response time for class k request over all sites is:

$$R_k = D_k + \sum_i \frac{\widehat{\Lambda}_k^i R_k^i}{\sum_j \widehat{\Lambda}_k^j} \tag{2}$$

where D_k denotes the queueing network delay (see Figure 3). After some elementary algebra, the CA problem can be formulated as:

(CA)

$$\min_{N_k^i, M_k^i} \sum_k \sum_i \overline{c}^i\,N_k^i + \widetilde{c}^i\,M_k^i$$

subject to

$$\widehat{\Lambda}_k^i < C^i\,\mu_k\,(N_k^i + M_k^i) \qquad \forall k \in K, \forall i \in I \tag{3}$$

$$\sum_i \frac{\widehat{\Lambda}_k^i\,(N_k^i + M_k^i)}{C^i\,\mu_k\,(N_k^i + M_k^i) - \widehat{\Lambda}_k^i} \leq (\overline{R}_k - D_k) \sum_j \widehat{\Lambda}_k^j \qquad \forall k \in K \tag{4}$$

$$\sum_{k \in K} N_k^i \leq \overline{N}^i \qquad \forall i \in I \tag{5}$$

$$N_k^i, M_k^i \geq 0$$

where constraints (5) guarantee that the number of VMs allocated to the whole set of classes at site i is at most equal to the number of *flat* VMs available at each site. Note that, in the problem formulation we have not imposed variables N_k^i and M_k^i to be integer, as in reality they are. Integer variables make the solution much more difficult because of the non-linear constraints (4). We therefore decide to deal with continuous variables, actually considering a relaxation of the real problem. However, if the optimal values of the variables are fractional and they are rounded to the closest integer solution, the gap between the solution of the real integer problem and the relaxed one is very small, justifying the use of a relaxed model. Furthermore, we can always choose a rounding to an integer solution which preserves the feasibility.

The CA problem has a linear objective function over a convex set. Indeed constraints (3) and (5) are

linear while the functions $\frac{\widehat{\Lambda}_k^i\,(N_k^i+M_k^i)}{C^i\,\mu_k\,(N_k^i+M_k^i)-\widehat{\Lambda}_k^i}$ are convex since their Hessian is of the form:

$$H = \frac{C^i\,\mu_k\,\left(\widehat{\Lambda}_k^i\right)^2}{(C^i\,\mu_k\,(N_k^i+M_k^i)-\widehat{\Lambda}_k^i)^3}\,\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}$$

whose eigenvalues are non-negative in the feasible set (in particular for condition (3)).

Hence, the global optimum solution can be obtained by solving (CA) in parallel at each site and by adopting standard non-linear solvers. This requires that each site broadcasts its $\widehat{\Lambda}_k^i$ predictions which can be obtained considering only local information. Since this broadcast is performed every T_1 time instants, the network overhead for the CA solution is negligible.

B. Load Redirect problem

Once the number of on demand instances has been determined, local requests can be dynamically redirected to other sites with time granularity T_2 in order to, e.g., avoid episodic local congestions due to the variability of the incoming workload at time granularity T_2 around its average hourly predicted value (see Figure 2).

According to equation (1), the average response time for executing locally WS class k at site i (i.e., without considering the network delay due to redirects) is given by:

$$\widehat{R}_k^i = D_k + \frac{1}{C^i\,\mu_k - \frac{\left(x_k^i+\sum_{j\neq i}\frac{g_k^{j,i}\,z_k^j}{G_k^j}\right)}{N_k^i+M_k^i}}.$$

During time interval T_2 the number of executions of class k requests at site i is $T_2\,x_k^i + T_2\sum_{j\neq i}\frac{g_k^{j,i}\,z_k^j}{G_k^j}$, and the average response time for remote requests is given by both $\widehat{R}_k^i$ and the delay $\sum_{j\neq i}\frac{d_k^{j,i}\cdot\frac{g_k^{j,i}\,z_k^j}{G_k^j}}{\sum_{j\neq i}\frac{g_k^{j,i}\,z_k^j}{G_k^j}}$. Therefore the total response time for executing class k request at site i is:

$$R_k^i = \widehat{R}_k^i + \frac{\sum_{j\neq i}\frac{z_k^j}{G_k^j}}{\left(x_k^i+\sum_{j\neq i}\frac{g_k^{j,i}\,z_k^j}{G_k^j}\right)},$$

and the total response time for class k request over all sites is on average:

$$R_k = \sum_i \frac{\left(x_k^i+\sum_{j\neq i}\frac{g_k^{j,i}\,z_k^j}{G_k^j}\right)R_k^i}{\sum_j \widehat{\widehat{\Lambda}}_k^j}.$$

The goal of our load redirect scheme is to cooperatively minimize the request average response times. Formally the LR can be formulated as a constraint programming problem since $R_k \leq \overline{R}_k$ must hold and

the cost for request execution is determined by the CA solution and is not influenced by the LR decision variables. However, in order to provide an efficient distributed solution, in our LR problem formulation we consider the total requests response time as the metric to be minimized. Indeed, experimental results have shown, that introducing an objective function allows us to speed up the distributed algorithm convergence relying on standard non-linear solvers. The LR problem can be formulated as follows:

(LR)

$$\min_{x_k^i, z_k^i} \sum_k \sum_i \left[D_k \left(x_k^i + \sum_{j \neq i} \frac{g_k^{j,i}\, z_k^j}{G_k^j} \right) + \frac{(N_k^i + M_k^i)\left(x_k^i + \sum\limits_{j \neq i} \frac{g_k^{j,i}\, z_k^j}{G_k^j}\right)}{C^i\, \mu_k\, (N_k^i + M_k^i) - (x_k^i + \sum\limits_{j \neq i} \frac{g_k^{j,i}\, z_k^j}{G_k^j})} + \sum_{j \neq i} \frac{z_k^j}{G_k^j} \right]$$

subject to

$$x_k^i + z_k^i = \widehat{\widehat{\Lambda}}_k^i \qquad \forall k \in K, i \in I, \tag{6}$$

$$x_k^i + \sum_{j \neq i} \frac{g_k^{j,i}\, z_k^j}{G_k^j} < C^i\, \mu_k\, (N_k^i + M_k^i) \qquad \forall k \in K, i \in I, \tag{7}$$

$$x_k^i, z_k^i \geq 0 \qquad \forall k \in K, i \in I.$$

Constraints (6) ensure that the overall class k requests at node i are locally executed or are redirected toward the other sites, while constraints (7) guarantee that VMs saturation conditions are avoided.

(LR) defines a centralized load balancing problem: All the system information (i.e., the local incoming workload predictions $\widehat{\widehat{\Lambda}}_k^i$) has to be gathered and used to get the optimal workload balancing. However, for large scale cloud systems, this centralized load balancing scheme is unsuitable because the $\widehat{\widehat{\Lambda}}_k^i$ broadcast may add a significant network overhead in massively distributed systems (recall T_2 is around 5-10 minutes).

The main idea we adopted to obtain a fully decentralized algorithm is to exploit prediction techniques to estimate at each site i also the workload due to the request redirect from other sites. In other words, each term $\sum\limits_{j \neq i} \frac{g_k^{j,i}\, z_k^j}{G_k^j}$ is replaced by the request redirect estimation $\widehat{\widehat{\lambda}}_k^i$ which becomes a parameter of the optimization problem. The idea to predict the requests coming from other sites differs from traditional workload prediction approaches, since they are especially oriented to forecast the local incoming traffic [14], [47] or the local load conditions [21], [29], [43].

Note that, the prediction $\widehat{\widehat{\Lambda}}_k^i$ at time scale T_2 exhibits a noisier behaviour with respect to $\widehat{\Lambda}_k^i$ evaluated at time scale T_1. Furthermore, if $\widehat{\Lambda}_k^i < \widehat{\widehat{\Lambda}}_k^i$, the CA results in a system underprovisioned and load redirection have to be performed in order to satisfy the M/G/1 queue equilibrium condition and to guarantee that $R_k \leq \overline{R}_k$. As a consequence of $\widehat{\widehat{\Lambda}}_k^i$ noisy behaviour, the request redirection is performed occasionally and the nature of the corresponding workload is characterized by a spikes.

The statistical behaviour of $\widehat{\widehat{\lambda}}_k^i$ and the temporal constraint imposed by the application context are the two main elements that influence the choice of the prediction model used by the LR algorithm and will be described in Section V.

The last term in the objective function can be rewritten as:

$$\sum_k \sum_i \sum_{j \neq i} \frac{z_k^j}{G_k^j} = \sum_k \sum_i (|I| - 1) \frac{z_k^i}{G_k^i}$$

Then, the problem (LR) becomes:

$(LR2)$

$$\min_{x_k^i, z_k^i} \sum_k \sum_i \left[D_k \left(x_k^i + \widehat{\widehat{\lambda}}_k^i \right) + \frac{(N_k^i + M_k^i)\left(x_k^i + \widehat{\widehat{\lambda}}_k^i\right)}{C^i \mu_k (N_k^i + M_k^i) - (x_k^i + \widehat{\widehat{\lambda}}_k^i)} + (|I| - 1) \frac{z_k^i}{G_k^i} \right]$$

subject to

$$\begin{aligned} x_k^i + z_k^i &= \widehat{\widehat{\Lambda}}_k^i \qquad \forall k \in K, i \in I, \\ x_k^i + \widehat{\widehat{\lambda}}_k^i &< C^i \mu_k (N_k^i + M_k^i) \qquad \forall k \in K, i \in I, \\ x_k^i, z_k^i &\geq 0 \qquad \forall k \in K, i \in I. \end{aligned}$$

$(LR2)$ can be separated by dropping the constants in the objective function and omitting the variables k and i indexes:

$(LR2_k^i)$

$$\min_{x,z} \left[D\, x + \frac{(N + M)\left(x + \widehat{\widehat{\lambda}}\right)}{C\, \mu\, (N + M) - (x + \widehat{\widehat{\lambda}})} + (|I| - 1) \frac{z}{G} \right]$$

subject to

$$x + z = \widehat{\widehat{\Lambda}} \tag{8}$$

$$x + \widehat{\widehat{\lambda}} < C\, \mu\, (N + M) \tag{9}$$

$$x, z \geq 0 \tag{10}$$

Each problem $(LR2_k^i)$ can be solved independently at each site from local information without the need to exchange data with other sites. The objective function of $(LR2_k^i)$ is convex. Indeed the Hessian is:

$$H = \begin{bmatrix} \frac{2\, C\, \mu\, (N+M)^2}{\left(C\, \mu\, (N+M) - (x + \widehat{\widehat{\lambda}})\right)^3} & 0 \\ 0 & 0 \end{bmatrix} \tag{11}$$

and in the feasible set under condition (9) the eigenvalues are non-negative. Then, the global optimum can be identified by applying the Karush Kuhn Tucker (KKT) conditions, which are necessary and sufficient for optimality in convex problems.

The solution is obtained through the following theorem.

***Theorem** 1: The global optimum solution for problem $(LR2_k^i)$ can be obtained considerig the following cases:*

- *Case a*:

$$x = (N+M)\left(C\,\mu - \sqrt{\frac{C\,\mu}{\frac{(|I|-1)}{G} - D}}\right) - \widehat{\widehat{\lambda}} \tag{12}$$

$$z = \widehat{\widehat{\Lambda}} + \widehat{\widehat{\lambda}} - \left(C\,\mu - \sqrt{\frac{C\,\mu}{\frac{(|I|-1)}{G} - D}}\right) \tag{13}$$

- *Case b*: $x = \widehat{\widehat{\Lambda}}$, $z = 0$, under the condition: $\widehat{\widehat{\Lambda}} + \widehat{\widehat{\lambda}} < C\,\mu\,(N+M)$
- *Case c*: $x = 0$, $z = \widehat{\widehat{\Lambda}}$, under the condition: $\widehat{\widehat{\lambda}} < C\,\mu\,(N+M)$

Proof. See Appendix A.

The global optimum can be obtained by inspection, evaluating the objective function for the three cases reported above. *Case a* corresponds to heavy load conditions for the system where the locally executed and redirected workloads can be obtained by (12) and (13). *Case b* corresponds to light load conditions where the overall load (local and redirected from other sites) is executed locally. Finally, in *Case c* the site serves only requests coming from other sites, while the whole local incoming workload is redirected to other sites. This last solution corresponds to the unusual situation arising when the site is under-provisioned and/or the delay to redirect requests to other site is very low.

V. Workload Prediction Models

For each site i and WS class k, our CA and LR algorithms use a prediction of the (real) local incoming workload Λ_k^i coming from the time zone where the site is located and also an estimate of the workload due to the request redirect from other sites $\lambda_k^i = \sum_{j \neq i} \frac{g_k^{j,i}\, z_k^j}{G_k^j}$. Λ_k^i is predicted at the two different time scales T_1 and T_2 in order to respect the temporal constraint imposed by the CA and LR problem, respectively. An estimate of λ_k^i is required only at time scale T_2.

The workload characterization literature is especially oriented to the analysis of incoming workload models and their statistical properties (e.g., [38], [14]), but it does not deal with the complexity and mostly unknown statistics of the redirected requests λ_k^i. However, in our setting the request redirect estimation could provide a simplification and an useful information for the solution of the LR problem.

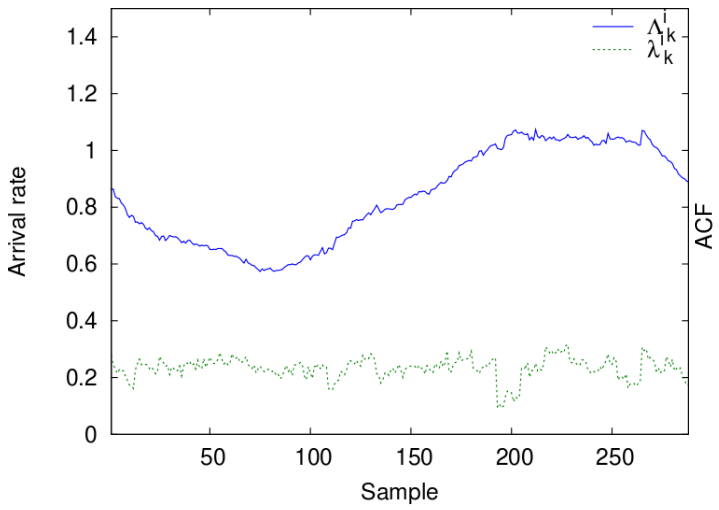

Fig. 4. Daily behaviour of Λ_k^i and λ_k^i.

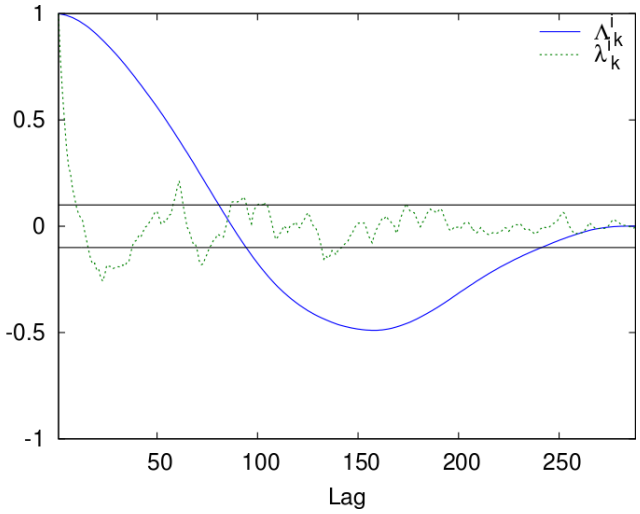

Fig. 5. Auto-Correlation function of real incoming workload.

Data reaching the site represent the historical basis for prediction and estimation models. Each historical information referring to one workload can be considered as a time series. An important premise is that no prediction model can work if the analyzed time series does not exhibit some predictability characteristics that is, some temporal dependency. The *auto-correlation analysis* on the time series allows us to show the presence of time dependence and to distinguish between the possibility of achieving or not prediction. Figure 4 reports an example of the daily behavior that characterizes the incoming workload Λ_k^i and the redirect workload from other sites λ_k^i. The latter has been obtained by solving several instances of problem centralized formulation of the LR problem under realistic conditions. The considered workloads exhibit two different behaviors. In particular, Λ_k^i is affected by fluctuations because of diurnal activity. Its high values of the autocorrelation function shown in Figure 5 confirm that the measures of the incoming workload are correlated. On the other hand, λ_k^i shows jittery perturbations and spikes varying in time and intensity that cause a quick decay of the ACF values shown in Figure 5. A high value of the auto-correlation function suggests that a past measure may be used to predict the future values with some degree of accuracy. The vice versa is true when the ACF between two points tends to zero. It is worth to point out that the results shown in Figure 5 are representative of a very large set of analyses we performed by considering the incoming workload to real systems (including an e-commerce site, an on-line banking site, and a large Italian University Web system).

The lesson learned from our results is the following. Independently of the cloud site, the slow decay of the auto-correlation function of the incoming workload Λ_k^i confirms its predictability, while the low

auto-correlation of λ_k^i evidences its unpredictable nature. For this reason, a reliable prediction of λ_k^i using only its past values is unfeasible. To overcome this issue and to provide a reliable representation of λ_k^i without the side effects of spikes, we estimate its actual value by using a filtering technique based on the Exponential Weighted Moving Average (EWMA), as suggested in [8]. In this way, we are able to reduce the out of scale value caused by instantaneous spikes and to extract a smoothed representation of the redirected workload.

In particular the estimate $\widehat{\widehat{\lambda}}_k^i(t)$ at time t is evaluated as follows:

$$\widehat{\widehat{\lambda}}_k^i(t) = \alpha\lambda_k^i(t) + (1-\alpha)\widehat{\widehat{\lambda}}_k^i(t-1) \tag{14}$$

where $0 < \alpha \leq 1$ is typically set to $1/(1+2\pi * f)$ and f is the cutoff frequency of the EWMA filter. See [39] for further details.

Concerning the prediction of Λ_k^i we rely on the Exponential Smoothing (ES) method which has been adopted in many fields for decades [23], [8] and is suitable to run-time and non-stationary applications. In general, each prediction mechanism is characterized by several alternative implementations, where the choice about filtering or not filtering input data, and choosing the best parameters of the model in a static or dynamic way are the most significant [22]. These alternatives characterize every prediction models and are especially important when they are used in an autonomic context without the possibility of human intervention and interpretation. In order to predict the local arrival rate Λ_k^i the choice for the ES models is motivated by the application context characterized by short-time predictions suitable for fast and autonomic decisions subject to real-time constraints of cloud systems. ES is an intuitive forecasting method that unequally weights the samples of the input time series Λ_k^i [32]. Non-uniform weighting is achieved through smoothing parameters which determine how much importance is assigned to each sample. In this paper, we consider a version of ES where parameters are dynamically chosen in order to adapt the prediction model to the workload fluctuations that characterize modern clouds [59].

We limit the discussion to the prediction of the arrival rate Λ_k^i at time scale T_2. The ES-prediction model at time scale T_1 can be obtained in the same way.

At sample t, the ES model estimates the local arrival rate at T_2 steps ahead, $\widehat{\widehat{\Lambda}}_k^i(t+T_2)$ as a weighted average of the last sample $\Lambda_k^i(t)$ and of corresponding predicted sample $\widehat{\widehat{\Lambda}}_k^i(t)$, that is equal to:

$$\widehat{\widehat{\Lambda}}_k^i(T_2) = \frac{1}{T_2}\sum_{t=1}^{T_2}\Lambda_k^i(t)$$

$$\widehat{\widehat{\Lambda}}_k^i(t+T_2) = \gamma_k^i(t)\widehat{\widehat{\Lambda}}_k^i(t) + (1-\gamma_k^i(t))\Lambda_k^i(t), \qquad t > T_2$$

where $\widehat{\widehat{\Lambda}}_k^i(T_2)$ is the initial predicted value and $0 < \gamma_k^i(t) < 1$ is the *smoothing factor* at current sample t related to the site i and to the class k that determines how much weight is assigned to each sample. We obtain a dynamic ES model by re-evaluating the smoothing factor $\gamma_k^i(t)$ at each prediction sample t. There are different proposals for the dynamic estimation of $\gamma_k^i(t)$ (e.g., [54], [56], [32], [26]). A widely used procedure is proposed by Trigg and Leach [54] that define the smoothing parameter as the absolute value of the ratio of the smoothed error, $A_k^i(t)$, to the absolute error, $E_k^i(t)$:

$$\gamma_k^i(t) = \frac{A_k^i(t)}{E_k^i(t)}$$

The smoothed and absolute errors are equal to:

$$A_k^i(t) = \phi\epsilon_k^i(t) + (1-\phi)A_k^i(t-T_2)$$
$$E_k^i(t) = \phi|\epsilon_k^i(t)| + (1-\phi)E_k^i(t-T_2)$$

where $\epsilon_k^i(t)$ is the forecast error at sample t, $\epsilon_k^i(t) = \Lambda_k^i(t) - \widehat{\widehat{\Lambda}}_k^i(t)$, and ϕ is set arbitrary with 0.2 being a common choice [54]. This dynamical choice of $\gamma_k^i(t)$ improves the prediction quality and limits the delay problem related to the traditional ES model based on a static choice of the γ_k^i parameter. We use an analogous implementation of the ES prediction model to predict the local arrival rate at T_1 steps ahead, $\widehat{\Lambda}_k^i$. For the sake of simplicity, in the remainder of the paper the t sample index will be omitted.

VI. Experimental Results

The proposed resource management algorithms have been evaluated for a variety of system and workload configurations. Section VI-A presents the experimental settings and the results on the scalability of our algorithms. Section VI-B presents a cost-benefit evaluation of our solution compared with other heuristics and state-of-the-art techniques [23], [59], [58]. Finally, Section VI-C shows the results of the application of our resource management algorithms in a real prototype environment deployed in Amazon EC2.

A. Algorithm Performance

To evaluate the efficiency of the proposed algorithms, we use a large set of randomly generated instances. All tests are performed on VMWare virtual machine based on Ubuntu 9.10 server running on an Intel Nehalem dual socket quad-core system with 32 GB of RAM. The virtual machine has a physical core dedicated with guaranteed performance and 4 GB of memory reserved. We use SNOPT 7.2.4 as the non-linear solver [36].

$\|K\|, \|I\|$	**Time**	$\|K\|, \|I\|$	**Time**	$\|K\|, \|I\|$	**Time**
100,20	2.3	500,20	29.3	1000,20	34.5
100,40	6.4	500,40	33.3	1000,40	98.6
100,60	9.3	500,60	65.6	1000,60	160.9

TABLE II

CAPACITY ALLOCATION PROBLEM SOLUTION EXECUTION TIME (SEC).

The number of cloud sites $|I|$ has varied between 20 and 60, the number of request classes $|K|$ between 100 and 1000. The maximum service rate of a capacity one VM for executing class k requests, μ_k, has been varied uniformly in (0.1,1), as in [3], while we set $\overline{R}_k = 3/\mu_k$, as in [11], [3], [12].

Table II reports, for problem instances of different sizes, the computational time in seconds required to solve the CA problem (figures are the means computed on ten different runs) demonstrating that our solution can determine the global optimal number of VM instances in few minutes, hence it is suitable to be applied on a hourly basis.

The computational time for solving a single $(LR2_k^i)$ problem instance is few milliseconds. Overall the LR solution is largely scalable and the load redirect mechanism can be reasonably applied at 5-10 minutes time scales without introducing any bottleneck.

B. Comparison with alternative literature proposals

In order to evaluate the quality of our approach we compare it against the solution which can be obtained by an oracle who has a perfect knowledge of the future workloads and performs capacity allocation and request redirection of time period T_1 and T_2. Furthermore, we perform a cost-benefit evaluation of our technique by considering other heuristics with a twofold aim. On the one hand we compare our solution with other state-of-the-art proposals which exploit the utilization principle and determine the number of VM instances according to an utilization threshold upper bound [23], [59], [58], [4]. On the other hand, we evaluate the effectiveness of the LR mechanism in the cloud. Indeed, in cloud systems the resource provisioning can be performed in very few minutes and hence instead of redirecting the load to other sites one can argue that the allocation of additional VMs to manage peak of traffics could be more effective. In this Section we report the results of the comparison of our CA and LR mechanisms with a set of solutions which perform a more fine grained CA at multiple time scales. In the remainder of this Section the following alternative solutions will be considered:

- *Oracle*: The CA is performed every hour, while th LR time period is 5 minutes. The workload prediction is 100% accurate, i.e., for any time instant $\widehat{\Lambda}_k^i = \Lambda_k^i$ and $\widehat{\widehat{\Lambda}}_k^i = \Lambda_k^i$.
- *Heuristic 1:* The CA is performed on a 5 minutes time horizon and the number of VMs is determined according to utilization thresholds as in other approaches proposed in the literature [23], [59], [58] and currently implemented also by IaaS providers (see, e.g., the very recent release of Amazon AWS Elastic Beanstalk [5]). In the evaluation, a life span of one hour for each instantiated VM has been considered. The number of VMs is determined such that the utilization of the VMs is equal to a given threshold τ_1. The VM provisioning is further triggered if the prediction of the VMs utilization is higher than a second threshold $\tau_2 > \tau_1$. Multiple analyses have been performed by adopting different thresholds: (τ_1, τ_2) = $(40\%, 50\%)$, $(50\%, 60\%)$, and $(60\%, 80\%)$.
- *Heuristic 2:* Same as *Heuristic 1* but the number of VMs is determined by optimally solving the CA problem reported in Section IV-A every 5 minutes.
- *Heuristic 3:* Same as *Heuristic 2* but with a 10 minutes time horizon.

The performance parameters of the request classes have been randomly generated as in Section VI-A. The local incoming workload is obtained from the traces, at 5 minutes sample time interval, related to a large dynamic Web-based system. We consider three scenarios:

- *Normal day scenario:* It describes the baseline workload where the number of clients requests changes by following the bi-modal profile shown in [15].
- *Heavy day scenario:* It exhibits a 40% uniform increment in the number of client requests with respect to the baseline workload.
- *Noisy day scenario:* It adds a noise component to the heavy day scenario. We added a white noise with zero mean and standard deviation equal to 10% of the heavy day peak. In this way, we test the robustness of the overall solution in highly variable contexts.

All scenarios are representative of the typical Web-based workload that is characterized by heavy-tailed distributions [25], [13]. Moreover, the heavy scenarios add burst arrivals and flash crowds [38] that contribute to augment the request skew, and they represent a more stressful testbed for prediction models. The motivation behind this choice is to demonstrate that our prediction algorithm works even in critical scenarios and our CA+LR mechanism are robust to workload variability, although the toughest goal of predicting hot spot events remains an open issue beyond the scope of this paper. In particular, the prediction model considered in this paper is able to provide an accurate prediction quality that, in terms of mean square error [22], is always lower than 10%.

Overall we have considered 12 sites. In our LR solution, the decision are made according to the prediction $\widehat{\widehat{\Lambda}}_k^i$ which allows to determine the overall traffic $x_k^i + \sum_{j \neq i} \frac{g^{j,i} z_k^j}{G^j}$ to be executed locally. This load includes two parts: the portion of the local incoming workload $x_k^i \leq \widehat{\widehat{\Lambda}}_k^i$, and the one due to the other sites redirect $\sum_{j \neq i} \frac{g^{j,i} z_k^j}{G^j}$. In the evaluation we consider as the local workload to the cloud sites the real workload Λ_k^i and the load actually redirected to other site is the traffic exceeding the prediction based value $x_k^i + \sum_{j \neq i} \frac{g^{j,i} z_k^j}{G^j}$. In the following quantitative analysis in our solution we set $T_1 = 1$ hour, and $T_2 = 5$ minutes.

Figures 6-11 plot the VM instantaneous costs over the 24 hours for the normal, heavy, and noisy day scenarios. Table III reports the percentage savings of our approach with respect to other solutions considering the total costs over the whole day. In particular Figures 6-8 compare our solution with Heuristic 1 for different values of the thresholds (τ_1, τ_2), while Figures 9-11 compare our approach with the oracle and Heuristics 2 and 3.

Results show that our approach is always the most convenient one and it is very close to the oracle solution. Furthermore, even if the oracle has a perfect knowledge of the future it does not always lead to the minimum instantaneous cost. Indeed, if the prediction is slightly lower then the real traffic, then our solution is cheaper than the oracle since a lower number of VMs is adopted. On the other hand, the oracle has the advantage of having no chance to violate the SLA (since there are no unexpected situations).

Concerning this latter issue, Figures 12-13 reports, as an example, the plot of the ratio $\frac{R_k}{\overline{R}_k}$ of the average response time with respect to the response time threshold of a class considered as a reference at site 1. The plot shape is pretty general and is independent of the considered site and request class. As the results show, the Heuristic 1 is very sensitive to the adopted thresholds. The $(40\%, 50\%)$ case is very conservative: it is around 35% more expensive than our approach but it always satisfies the response time threshold (the ratio is strictly lower than 1). Vice versa, the $(60\%, 80\%)$ case has costs close to our solution (only 2-4% higher) but it introduces a very large number of SLA violations especially in the noisy day scenario (see Figure 13). Our solution introduces overall only 37 violations over the 3456 time intervals considered in the 12 sites, over the whole day. Furthermore, Heuristics 1 is more sensitive to traffic variability (the costs incurred in the noisy day scenario are higher). Heuristics 2 and 3 perform better than Heuristics 1, since the number of VMs is optimally determined by the CA problem solutions. However, the LR mechanism is still effective since allows to reduce costs by 4-12%. The fine grained resource allocation introduced by Heuristics 2 and 3 indeed ends into an over-provisioning and better performance, while the LR mechanism allows to forward traffic spikes to other locations without

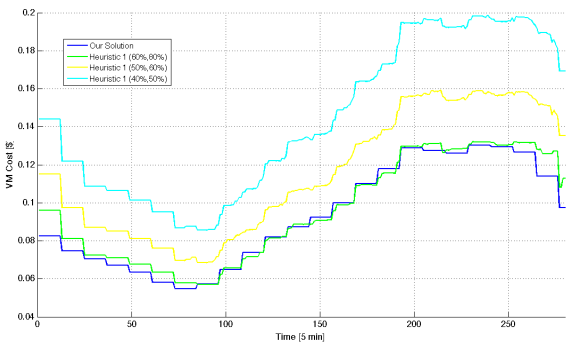

Fig. 6. VM instances costs for the normal day scenario.

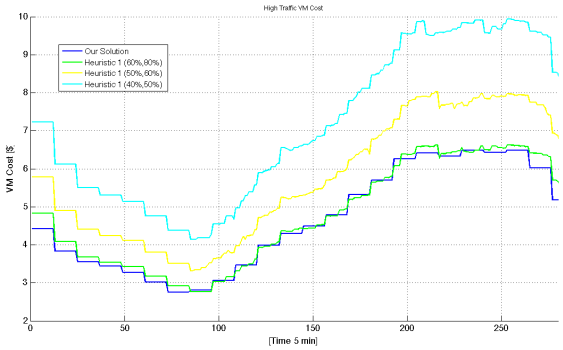

Fig. 7. VM instances costs for the heavy day scenario.

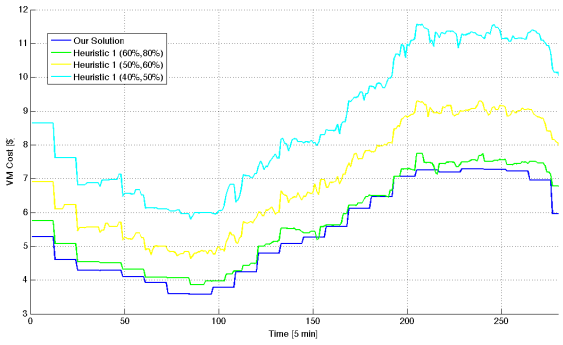

Fig. 8. VM instances costs for the noisy day scenario.

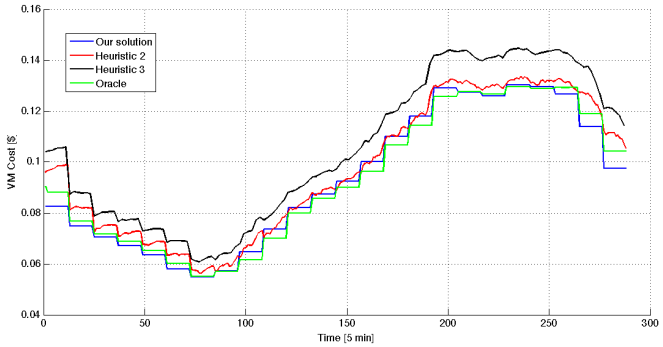

Fig. 9. VM instances costs for the normal day scenario.

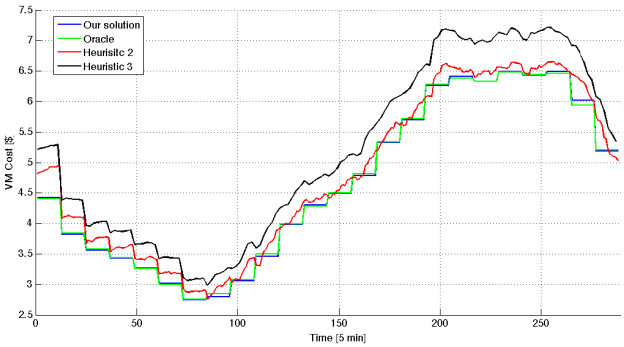

Fig. 10. VM instances costs for the heavy day scenario.

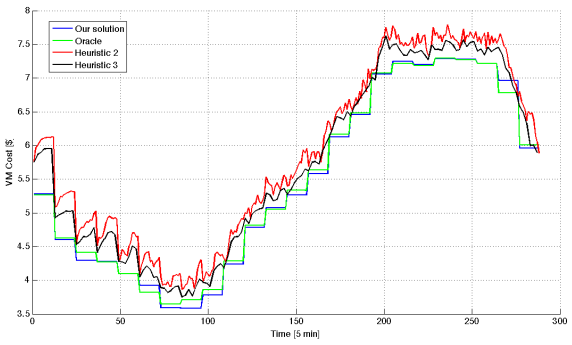

Fig. 11. VM instances costs for the noisy day scenario.

overcoming in any additional capacity allocation or significant SLA violations.

Alternative solution	% Savings		
	Normal day	Heavy day	Noisy day
Oracle	0.00	0.00	0.00
Heuristic 1 - $(40\%, 50\%)$	35.47	34.86	36.84
Heuristic 1 - $(50\%, 60\%)$	19.53	18.83	21.40
Heuristic 1 - $(60\%, 80\%)$	3.12	2.25	4.93
Heuristic 2	4.30	3.26	4.44
Heuristic 3	11.56	10.27	6.98

TABLE III

VM PERCENTAGE COST SAVINGS OVER THE 24 HOURS OBTAINED BY OUR APPROACH.

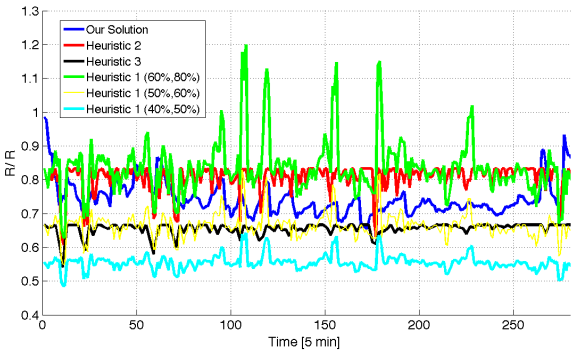

Fig. 12. Response time threshold ratio for a reference class, normal day scenario.

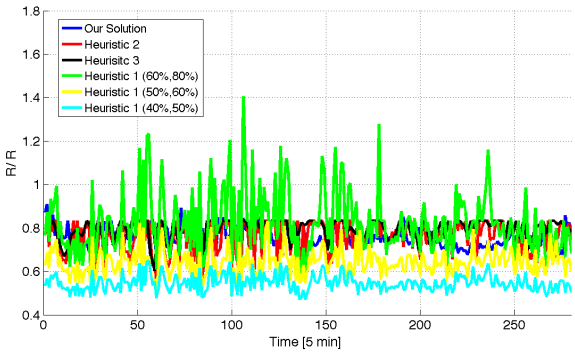

Fig. 13. Response time threshold ratio for a reference class, noisy day scenario.

C. Amazon EC2 Test

The effectiveness of our resource management algorithms has been also evaluated on a real prototype environment deployed on Amazon EC2. We performed experiments running the JSP implementation of the SPECweb2005 [52] benchmark. In particular, we have considered the *banking* workload, which simulates the access to an on line banking Web site implementing a full HTTPS load. SPECweb2005 includes four components: The load generators, the client coordinator, the Web server, and the back-end simulator. The SPECweb2005 load generators inject workload to the system according to a closed model. Users sessions are started according to a given number of users who continuously send requests for dynamic Web pages, wait for an average think time $Z = 10$s, and then access another page or leave the system according to

a pre-defined session profile[1]. The client coordinator initializes all the other systems, monitors the test, and collects the results. The Web server is the component target of the performance assessment (Apache Tomcat 5.5.27 in our setup), while the back-end simulator emulates the database and application parts of the benchmark and it is used to determine the dynamic content of the Web pages.

The Web server has been deployed on a large instance, while the load generators, the client coordinator, and the back-end simulator have been hosted by extra-large Amazon instances (in this way we are guaranteed that they are not the system bottleneck). The test is performed by deploying VM instances in Virginia (Site 1), North California (Site 2), and Europe (Site 3) Amazon regions. We have obtained an estimate of the maximum service rate parameters and the network delay among different Amazon sites by performing an extensive off line profiling along the lines of [46], [40] by collecting a set of statistics and minimizing the mean square error for the response time. Figure 14 shows how the adopted performance model fits the measured data, where $D = 0.66$ sec. and $\mu = 49.02$ requests/sec. The maximum percentage error on the average response time estimation is less than 20%. The network transfer delay estimated for the three sites are reported in Table IV.

Network Transfer Delay	**Value [sec]**
$d^{1,2} \simeq d^{2,1}$	0.20
$d^{1,3} \simeq d^{3,1}$	0.29
$d^{2,3} \simeq d^{3,2}$	0.40

TABLE IV

NETWORK TRANSFER DELAY ESTIMATED AMONG EC2 AMAZON REGIONS.

We set $\overline{R} = 0.6$ seconds as the threshold for the average response time and the overall test lasts two hours. We have generated an appropriate traffic profile (reported in Figures 15-17). In particular Site 1 is characterized by a fluctuating workload, while at Sites 2 and 3 the workload has an almost linear trend (with some noise superimposed). We run the CA algorithm twice at the beginning of each hour. The LR algorithm is run every 10 minutes. The number of on demand VMs allocated at each site for the two hours is reported in Table V. In every region, the load is evenly shared among multiple instances at

[1]Our optimization framework is based on an open performance model: We have estimated the overall incoming workload a priori as $\Lambda_k = N_k/Z$, since in the considered number of users range, VMs response time was significantly lower than the user think time (we recall that, for the response time law, $N_k = (R_k + Z_k) \cdot \Lambda_k$).

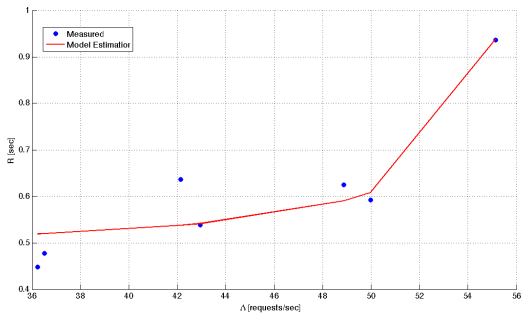

Fig. 14. Performance Model Data Fitting.

each site by registering the VMs with an Amazon Elastic Load Balancer [4]. The load is redirected for 6 over the 12 time intervals and, sometimes, one site concurrently executes the workload coming from the other two regions. Table VI details the nature of the workload executed at each site for every time interval. The overall incoming workload to North California is higher than the others, especially during the second hour. As a consequence, the traffic redirection from Site 2 towards the others is performed more frequently.

Hour	**Site 1**	**Site 2**	**Site 3**
1	2	3	2
2	2	4	2

TABLE V

NUMBER OF VMS ALLOCATED AT EACH AMAZON REGION.

Figures 18-20 show the overall traffic served at the three sites. Finally, Figure 21 reports the average response time (evaluated every 10 seconds) and shows that our CA+LR algorithms are effective since the system satisfy SLA always but in two cases, and it is able to react to abrupt workload variations.

VII. RELATED WORK

With the development of autonomic computing systems, dynamic resource allocation techniques have received a great interest both within the Industry and Academia. The solutions proposed can be classified in centralized and distributed. In a centralized approach, a dedicated entity is in charge of establishing

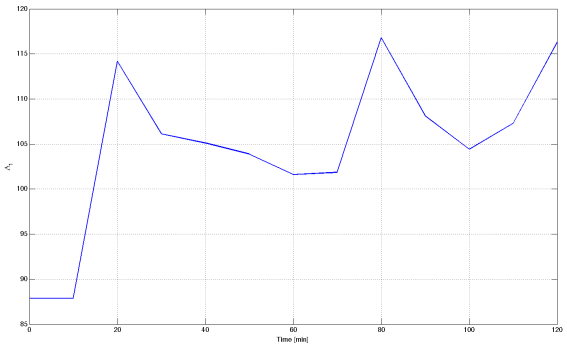

Fig. 15. Local Incoming Workload at Virginia EC2 Site.

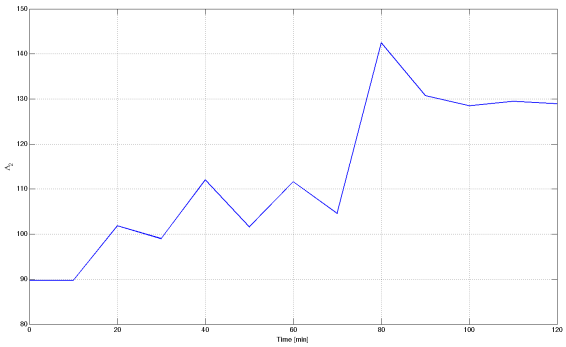

Fig. 16. Local Incoming Workload at North California EC2 Site.

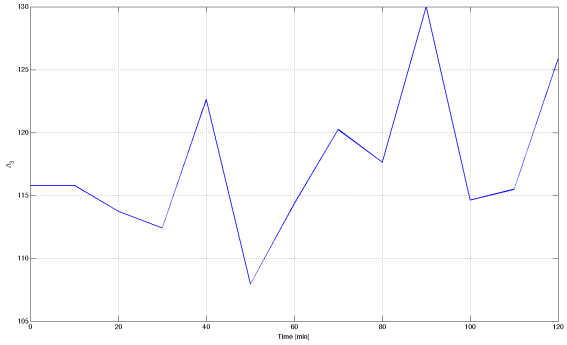

Fig. 17. Local Incoming Workload at Europe EC2 Site.

resource allocation, admission control or load balancing for the autonomic system and has a global knowledge of the resources state in the whole network [12], [55].

Centralized solutions are not suitable for geographically distributed systems, such as the cloud or more in general massively distributed systems [7], [34], [31], since no one entity has global information about all system resources. Indeed, the communication overhead required to share the resource state information is not negligible and the delay to achieve state information from remote nodes could lead a centralized resource manager to very inaccurate decisions due to dynamic changes in system conditions, as well as resource consumption fluctuations or unexpected events [34].

Distributed resource management policies have been proposed to govern efficiently geographically distributed systems that cannot implement centralized decisions and support strong interactions among the remote nodes [7]. Distributed resource management is very challenging, since one node's decisions may inadvertently degrade the performance of the overall system, even if they greedily optimize the

LR Time Instant	Site 1	Site 2	Site 3
1	Local	Local	Local
2	Local	Local + Site 1	Local + Site 1
3	Local	Local	Local
4	Local + Site 2 + Site 3	Local + Site 3	Local + Site 2
5	Local	Local	Local
6	Local + Site 2	Local	Local + Site 2
7	Local	Local	Local
8	Local + Site 2	Local + Site 1	Local + Site 1 + Site 2
9	Local + Site 3	Local + Site 3	Local
10	Local	Local	Local
11	Local	Local	Local
12	Local	Local + Site 1	Local + Site 1

TABLE VI

WORKLOAD EXECUTED AT EACH AMAZON REGION.

performance of its nodes. Sometimes, local decisions could lead the system even to unstable oscillations [33]. It is difficult to determine the best control mechanism at each node in isolation, so that the overall system performance is optimized. Dynamically choosing when, where and how allocate resources and coordinating the resource allocation accordingly is an open problem and is becoming more relevant with the advances of clouds [31].

One of the first contributions for resource management in geographically distributed systems has been proposed in [7], where novel autonomic distributed load balancing algorithms have been proposed. The algorithms aim to reduce the communication overhead among the system components and to gradually shift portions of requests among the distributed nodes. Authors have proposed a trend-based activation scheme that is based on local system information and exploit cooperation among other nodes. In the work [49] mechanisms to optimize performance within a geographical node and to redirect requests to the best remote node have been proposed. In distributed streaming networks, authors in [34] have proposed a joint admission control and resource allocation scheme, while [57] has proposed optimal scheduling techniques. Scheduling in streaming systems faces the problem that the incoming workload would far exceed system capacity much of the time. In cloud based systems, a joint solution for the capacity allocation and load balancing among multiple IaaS sites based on Lagrangian decomposition techniques has been presented

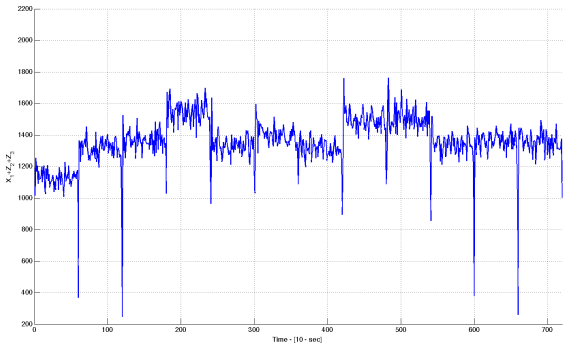

Fig. 18. Overall traffic served at Virginia EC2 site.

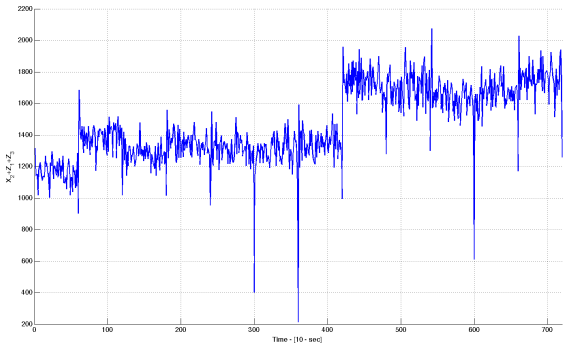

Fig. 19. Overall traffic served at North California EC2 site.

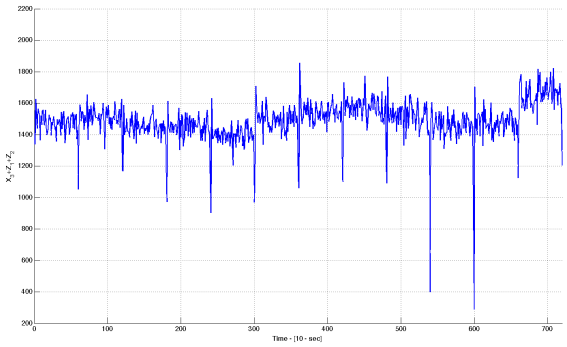

Fig. 20. Overall traffic served at Europe EC2 site.

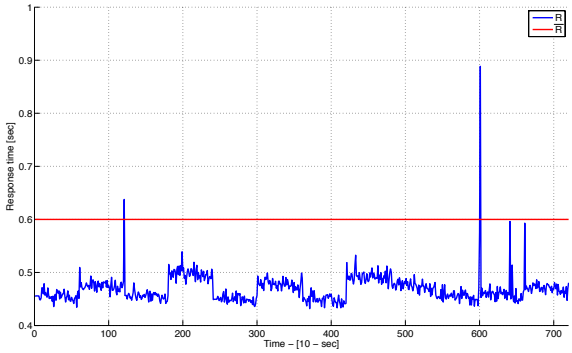

Fig. 21. Average response time measured for the SPECweb2005 banking workload.

in [11], [10], while in [48] a structured peer-to-peer network based on distributed hash tables has been proposed supporting service discovery, self-managing, and load-balancing of cloud based applications.

In the distributed resource management area, researches are borrowing some ideas also from the biological world [50]. Biology-inspired techniques have been used in self-aggregation algorithms to establish and maintain groups of software components that cooperate to reach a common goal [27], [45], to implement performance and energy-aware virtual machines live migration, and for the self-provisioning of cloud based applications [19]. To the best of our knowledge, this paper is the first contribution that proposes an analytical solution to the capacity allocation and load redirect mechanism for cloud systems.

With the focus on load redirect, a considerable amount of work has been done facing the problem of sharing the load evenly in massively distributed Web systems [24]. Two-level dispatching schemes, where client requests are initially assigned by the DNS, and each Web server may redirect a request to any

other server of the system through the HTTP redirection mechanism, have been proposed in [9], [20]. DNS-based routing is the first solution that has been proposed to handle multiple Web servers hosting a Web site. It was originally conceived for locally distributed Web systems even if now it is commonly used in geographically distributed Web systems [41]. Other request redirection strategies that use the built-in HTTP mechanism have been proposed in [17], [35]. Finally, in [42] the URL rewriting mechanism have been proposed. Such a mechanism integrated with a multiple-level DNS routing technique is also used by Content Delivery Networks, such as Akamai [2].

Prediction algorithms have also an important role in the development of autonomic computing systems. A recent survey can be found in [22]. Prediction in a noisy context related to autonomic decisions does not require just the application of one suitable model, but a complex mechanism with several alternatives. The presence of dozen of models, each with many parameters, disorients the choice because, to the best of our knowledge, there is no criteria for choosing which model is better in a certain context, how to set the model parameters, whether input data treatment is really necessary. For example, several models were proposed to forecast the resource state of servers that can be extended to Internet data center contexts. For example, the Linear Regression model was applied to Web-based systems [21], the Exponential Smoothing to the running time of jobs [30], [51], the Holt's methods to the throughput of large TCP transfers [37], the Autoregressive and Autoregressive Integrated Moving Average to the load [29], [53] and network traffic [47]; the tendency based predictors, such as the Cubic Spline, to the CPU load [43].

VIII. Conclusion

In this paper distributed Capacity Allocation and Load Redirect algorithms have been proposed aiming at the minimization of costs in infrastructure as a service cloud systems. The presented solution acts at multiple time scales and integrates prediction techniques with non-linear optimization models. A sound decomposition of the optimization problems has been provided. The effectiveness of our approach has been assessed by performing simulation and experiments in a real prototype environment running in Amazon EC2. Synthetic as well as realistic workloads and a number of different scenarios of interest have been considered. A comparison with other solutions proposed in the literature shows that our solutions outperform alternative methods providing costs up to 35% lower, without introducing significant SLA violations. Furthermore, our solutions are very closed to the ones found by an oracle with perfect knowledge of the future. Ongoing work aims at extending the optimization problem in order to support also the decision on the capacity of the running instances and to model the use of private infrastructures.

ACKNOWLEDGEMENT

The experimentation on Amazon EC2 has been supported by Amazon AWS in Education research grant. The work of Danilo Ardagna and Barbara Panicucci has been partially supported by the GAME-IT research project funded by Politecnico di Milano and by the European Commission, Programme IDEAS- ERC, Project 227977-SMScom. Sara Casolari and Michele Colajanni acknowledge the support of MIUR-PRIN project AUTOSEC "Autonomic Security".

REFERENCES

[1] B. Abraham and J. Ledolter. *Statistical Methods for Forecasting*. John Wiley and Sons, 1983.

[2] Akamai. http://www.akamai.com.

[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 Elastic Cloud. http://aws.amazon.com/ec2/.

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

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

[7] M. Andreolini, S. Casolari, and M. Colajanni. Autonomic request management algorithms for geographically distributed internet-based systems. In *SASO*, 2008.

[8] M. Andreolini, S. Casolari, and M. Colajanni. Models and framework for supporting run-time decisions in web-based systems. *ACM Trans. on the Web*, 2(3), 2008.

[9] D. Andresen, T. Yanh, and O. H. Ibarra. Towards a scalable distributed www server on networked workstations. In *Journal of Parallel and Distributed Computing*, volume 42, pages 91–100, 1997.

[10] D. Ardagna, S. Casolari, and B. Panicucci. Flexible distributed capacity allocation and load redirect algorithms for cloud systems. In *Proc. of the 4th International Conference on Cloud Computing (IEEE Cloud 2011). To Appear*, 2011.

[11] D. Ardagna, C. Ghezzi, B. Panicucci, and M. Trubian. Service provisioning on the cloud: Distributed algorithms for joint capacity allocation and admission control. In *ServiceWave*, 2010.

[12] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-Aware Autonomic Resource Allocation in Multi-tier Virtualized Environments. *IEEE Trans. on Services Computing,* available on line.

[13] M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large Web-based shopping system. 1(1):44–69, Aug. 2001.

[14] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana. Predictability of Web server traffic congestion. In *Proc. of the 10th Workshop of Web Content Caching and Distribution*, Sophia Antipolis, FR, Sep. 2005.

[15] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and Y. Yimwadsana. Predictability of web server traffic congestion. In *WCW Proc.*, 2005.

[16] M. Bennani and D. Menascé. Resource Allocation for Autonomic Data Centers Using Analytic Performance Models. In *IEEE Int'l Conf. Autonomic Computing Proc.*, 2005.

[17] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol http/1.0. RFC 1945, May 1996.

[18] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing Networks and Markov Chains*. J. Wiley, 1998.

[19] A. B. Caprarescu, N. M. Calcavecchia, E. Di Nitto, and D. J. Dubois. Sos cloud: Self-organizing services in the cloud. In *BIONETICS '10: Proceedings of the 5th International Conference on Bio-Inspired Models of Network, Information and Computing Systems*, 2010.

[20] V. Cardellini, M. Colajanni, and P. Yu. Request redirection algorithms for distributed web systems. *Parallel and Distributed Systems, IEEE Transactions on*, 14(4):355 – 368, 2003.

[21] S. Casolari and M. Colajanni. Short-term prediction models for server management in internet-based contexts. *Elsevier - Decision Support Systems*, 48, 2009.

[22] S. Casolari and M. Colajanni. On the selection of models for runtime prediction of system resources. *Autonomic Systems, Springer (Eds. Danilo Ardagna, Li Zhang)*, 2010.

[23] L. Cherkasova and P. Phaal. Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites. *IEEE Transactions on Computers*, 51(6), June 2002.

[24] M. Colajanni, P. S. Yu, and V. Cardellini. Dynamic load balancing in geographically distributed heterogeneous web servers. In *ICDCS*, pages 295–302, 1998.

[25] M. E. Crovella, M. S. Taqqu, and A. Bestavros. Heavy-tailed probability distributions in the World Wide Web. In *A Practical Guide To Heavy Tails*, pages 3–26. Chapman and Hall, New York, 1998.

[26] J. Dennis. A performance test of a run-based adaptive exponential smoothing. *Production and Inventory Management*, 19, 1978.

[27] E. Di Nitto, D. Dubois, and R. Mirandola. Self-aggregation algorithms for autonomic systems. In *Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd*, 2007.

[28] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing*, 13(5):10–13, 2009.

[29] P. Dinda and D. O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4), Dec. 2000.

[30] M. Dobber, G. Koole, and R. Van det Mei. Dynamic load balancing experiments in a grid. In *Proc. of the 5th IEEE International Symposium on Cluster Computing and on the Grid*, May 2005.

[31] H. Erdogmus. Cloud computing: Does nirvana hide behind the nebula? *IEEE Softw.*, 26(2):4–6, 2009.

[32] S. Everette and J. Gardner. Exponential smoothing: State of the art. *Journal of Forecasting*, 4, 1985.

[33] P. Felber, T. Kaldewey, and S. Weiss. Proactive hot spot avoidance for web server dependability. *Reliable Distributed Systems, IEEE Symposium on*, pages 309–318, 2004.

[34] H. Feng, Z. Liu, C. H. Xia, and L. Zhang. Load shedding and distributed resource control of stream processing networks. *Perform. Eval.*, 64(9-12):1102–1120, 2007.

[35] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext transfer protocol http/1.1. RFC 2616, June 1999.

[36] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal of Optimization*, 12:979–1006, 2002.

[37] Q. He, C. Dovrolis, and M. Ammar. On the predictability of large transfer tcp throughput. In *Proc. of ACM SIGCOMM 2005*, Aug. 2005.

[38] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and Web sites. In *WWW2002 Proc.*, Honolulu, HW, May 2002.

[39] M. Kendall and J. Ord. *Time Series*. Oxford University Press, 1990.

[40] D. Kumar, A. N. Tantawi, and L. Zhang. Real-time performance modeling for adaptive software systems with multi-class workload. In *MASCOTS*, 2009.

[41] T. T. Kwan, R. E. McGrath, and D. A. Reed. Ncsa's world wide web server: Design and performance. *IEEE Computer*, 28(11):68–74, 1995.

[42] Q. Li and B. Moon. Distributed cooperative aspache web server. In *Proc. of 10th Int'l World Wide Web Conf.*, May 2001.

[43] Y. Lingyun, I. Foster, and J. M. Schopf. Homeostatic and tendency-based CPU load predictions. In *Proc. of the 17th Parallel and distributed processing Symp.*, Nice, FR, 2003.

[44] D. A. Menascé and V. Dubey. Utility-based QoS Brokering in Service Oriented Architectures. In *IEEE ICWS Proc.*, pages 422–430, 2007.

[45] E. D. Nitto, D. Dubois, R. Mirandola, F. Saffre, and R. Tateson. Self-aggregation techniques for load balancing in distributed systems. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, 2008.

[46] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. Cpu demand for web serving: Measurement analysis and dynamic estimation. *Perform. Eval.*, 65(6-7):531–553, 2008.

[47] Y. Qiao and P. Dinda. Network traffic analysis, classification, and prediction. Technical report, 2003.

[48] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar. Peer-to-peer cloud provisioning: Service discovery and load-balancing. In N. Antonopoulos and L. Gillam, editors, *Cloud Computing*, Computer Communications and Networks, pages 195–217. Springer London, 2010.

[49] S. Ranjan and E. Knightly. High-performance resource allocation and request redirection algorithms for web clusters. *IEEE Trans. Parallel Distrib. Syst.*, 19:1186–1200, September 2008.

[50] M. Shackleton and P. Marrow. Editorial, special issue on nature-inspired computation. BT Technology Journal, 2000.

[51] K. Shum. Adaptive distributed computing through competition. In *Proc. of the 3th IEEE International Conference on Configurable Distributed System*, May 1996.

[52] Spec. The SPECWeb2005 benchmark. http://www.spec.org/web2005/.

[53] N. Tran and D. Reed. Automatic ARIMA time series modeling for adaptive I/O prefetchingp. 15(4):362–377, Apr. 2004.

[54] D. Trigg and A. Leach. Exponential smoothing with an adaptive response rate. *Operational Research Quarterly*, 18, 1967.

[55] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi. Analytic modeling of multitier Internet applications. *ACM Transaction on Web*, 1(1), January 2007.

[56] D. Whybark. Comparison of adaptive forecasting techniques. *Logistics Transportation Review*, 8.

[57] J. L. Wolf, N. Bansal, K. Hildrum, S. Parekh, D. Rajan, R. Wagle, K.-L. Wu, and L. Fleischer. Soda: An optimizing scheduler for large-scale stream-based distributed computer systems. In *Middleware*, 2008.

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

[59] 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.

APPENDIX A

Theorem 1 Proof.

Let us introduce multipliers L_1 for constraint (8), L_2 for the queue equilibrium condition (9), and L_3, L_4 for the non negativity constraints on x and z respectively (L_1 is unrestricted in sign, while the other multipliers have to be non-negative).

The Lagrangian for $(LR2_k^i)$ problem is

$$\mathcal{L}(x,z,L_1,L_2,L_3,L_4) = D\,x + \frac{(N+M)\left(x+\widehat{\widehat{\lambda}}\right)}{C\,\mu\,(N+M)-(x+\widehat{\widehat{\lambda}})} + (|I|-1)\,\frac{z}{G}$$
$$-L_1\,(\widehat{\widehat{\Lambda}}-x-z) - L_2\,(C\,\mu\,(N+M)-x-\widehat{\widehat{\lambda}}) - L_3\,x - L_4\,z.$$

The KKT conditions provide the following set of equations to be solved together with the feasibility of x and z for problem $(LR2_k^i)$:

$$\begin{cases} \frac{\partial\mathcal{L}}{\partial x} = D + \frac{C\,\mu\,(N+M)^2}{\left(C\,\mu\,(N+M)-(x+\widehat{\widehat{\lambda}})\right)^2} + L_1 + L_2 - L_3 &= 0 \\ \frac{\partial\mathcal{L}}{\partial z} = \frac{(|I|-1)}{G} + L_1 - L_4 &= 0 \\ L_1\,(\widehat{\widehat{\Lambda}}-x-z) &= 0 \\ L_2\,(C\,\mu\,(N+M)-x-\widehat{\widehat{\lambda}}) &= 0 \\ L_3\,x &= 0 \\ L_4\,z &= 0 \end{cases}$$

First of all, note that we have always $L_2 = 0$, since the queue equilibrium condition is a strict inequality.

Let us now examine the possible cases:

- $x > 0,\ z > 0$: If we assume $x > 0$ and $z > 0$ then, for the non negativity slack conditions, we get $L_3 = L_4 = 0$. From the second KKT system equation we obtain $L_1 = -\frac{(|I|-1)}{G}$ and then we can derive equation (12). Finally from (8) we get *case a* equation (13).
- $x = \widehat{\widehat{\Lambda}}$: If we set $x = \widehat{\widehat{\Lambda}}$, then from equation (8) we get $z = 0$. The solution is feasible iff $\widehat{\widehat{\Lambda}} + \widehat{\widehat{\lambda}} < C\,\mu\,(N+M)$ holds. Hence, we get *case b* conditions.
- $z = \widehat{\widehat{\Lambda}}$: If we set $z = \widehat{\widehat{\Lambda}}$, from equation (8) we get $x = 0$, and hence the local incoming workload is totally redirected. The solution is feasible iff $\widehat{\widehat{\lambda}} < C\,\mu\,(N+M)$ holds. Hence, we get *case c* conditions.
- $x = 0,\ 0 \le z < \widehat{\widehat{\Lambda}}$ or $0 \le x < \widehat{\widehat{\Lambda}},\ z = 0$: Under these assumptions no solution can be obtained since equation (8) can not hold.