Generalized Nash Equilibria for the Service Provisioning Problem in Cloud Systems

Danilo Ardagna, Barbara Panicucci
Mauro Passacantando

Report n. 2011.27

Generalized Nash Equilibria for the Service Provisioning Problem in Cloud Systems

Danilo Ardagna*, Barbara Panicucci*, Mauro Passacantando**

*Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza Leonardo da Vinci 32, 20133 Milan, Italy. E-mail: {ardagna,panicucci}@elet.polimi.it.

**Università di Pisa Dipartimento di Matematica Applicata, Via Buonarroti 1/c, 56127 Pisa, Italy. E-mail: passacantando@dma.unipi.it.

Abstract

In recent years the evolution and the widespread adoption of virtualization, service-oriented architectures, autonomic, and utility computing have converged letting a new paradigm to emerge: The Cloud Computing. Clouds allow the on-demand delivering of software, hardware, and data as services. Currently the Cloud offer is becoming day by day wider since all the major IT Companies and Service providers, like Microsoft, Google, Amazon, HP, IBM, and VMWare have started providing solutions involving this new technological paradigm.

As Cloud-based services are more numerous and dynamic, the development of efficient service provisioning policies become increasingly challenging. In this paper we take the perspective of Software as a Service (SaaS) providers which host their applications at an Infrastructure as a Service (IaaS) provider. Each SaaS needs to comply with quality of service requirements, specified in Service Level Agreement (SLA) contracts with the end-users, which determine the revenues and penalties on the basis of the achieved performance level. SaaS providers want to maximize their revenues from SLAs, while minimizing the cost of use of resources supplied by the IaaS provider. Moreover, SaaS providers compete and bid for the use of infrastructural resources. On the other hand, the IaaS wants to maximize the revenues obtained providing virtualized resources.

In this paper we model the service provisioning problem as a generalized Nash game and we show the existence of generalized Nash equilibria. Moreover, we propose two solution methods based on the best-reply dynamics and we prove their convergence in a finite number of iterations to a generalized Nash equilibrium. In particular, we develop an efficient distributed algorithm for the run-time allocation of IaaS resources among competing SaaS providers. We demonstrate the effectiveness of our approach by simulation and performing tests on a real prototype environment deployed on Amazon EC2. Results show that, compared to other state-of-the-art solutions, our model can improve the efficiency of the Cloud system evaluated in term of Price of Anarchy by 50-70%.

I. Introduction

Cloud Computing has been a dominant IT news topic over the past few years. It is essentially a way for IT companies to deliver software/hardware on-demand as services through the Internet. Cloud computing applications are generally priced on a subscription model, so end-users may pay a yearly usage fee, for example, rather than the more familiar model of purchasing software licenses. The Cloud-based services are not only restricted to software applications (Software as a Service – SaaS), but could also be the platform for the deployment and execution of applications developed in house (Platform as a Service – PaaS) and the hardware infrastructure (Infrastructure as a Service – IaaS).

In the SaaS paradigm applications are available over the Web and provide Quality of Service (QoS) guarantees to end-users. The SaaS provider hosts both the application and the data, hence the end-user is able to use and access the service from all over the world. With PaaS, applications are developed and deployed on platforms transparently managed by the Cloud provider. The platform typically includes databases, middleware and also development tools. In IaaS systems, virtual computer environments are provided as services and servers, storage, and network equipment can be outsourced by customers without the expertise to operate them.

Many Companies, e.g. Google and Amazon, are offering Cloud computing services such as Google's App Engine and Amazon's Elastic Compute Cloud (EC2) or Microsoft Windows Azure. Large data centers provide the infrastructure behind the Cloud and virtualization technology makes Cloud computing resources more efficient and cost-effective both for providers and customers. Indeed, end-users obtain the benefits of the infrastructure without the need to implement and administer it directly adding or removing capacity almost instantaneously on a "pay-as-you-use" basis. Cloud providers can, on the other hand, maximize the utilization of their physical resources also obtaining economies of scale.

The development of efficient service provisioning policies is among the major issues in Cloud research. Indeed, modern Clouds live in an open world characterized by continuous changes which occur autonomously and unpredictably. In this context, game theoretic methods and approaches allows to gain an in-depth analytical understanding of the service provisioning problem. Game Theory has been successfully applied to diverse problems such as Internet pricing, flow and congestion control, routing, and networking [6], [39]. One of the most widely used "solution concept" in Game Theory is the Nash Equilibrium approach: A set of strategies for the players constitute a Nash Equilibrium if no player can benefit by changing his/her strategy while the other players keep their strategies unchanged or, in other words, every player is playing a *best response* to the strategy choices of his/her opponents.

In this paper we take the perspective of SaaS providers which host their applications at an IaaS provider. Each SaaS provider want to maximize its profit while complying with QoS requirements, specified in Service Level Agreement (SLA) contracts with the end-users, which determine the revenues and penalties on the basis of the achieved performance level. The profit of the SaaS is given by the revenues from SLAs minus the cost sustained for using the resources supplied by the IaaS. However, each SaaS behaves selfishly and competes with others SaaS for the use of infrastructural resources supplied by the IaaS. The IaaS, in his turn, wants to maximize the revenues obtained providing the resources. To capture the behavior of SaaSs and IaaS in this conflicting situation in which the best choice for one depends on the choices of the others, we recur to the *Generalized Nash Equilibrium* (GNE) concept [20], which is an extension of the classical Nash equilibrium [36]. In this paper the service provisioning problem will be modelled as a Generalized Nash game. We then use Game Theory results for developing efficient algorithms for the run-time management and allocation of IaaS resources to competing SaaSs suitable also for a *fully distributed* implementation. Multiple solutions achieving Generalized equilibria are

proposed and evaluated in terms of their efficiency with respect to the *social optimum* of the Cloud.

We demonstrate the effectiveness of our approach by simulation and performing tests on a real prototype environment. Extensive experiments show that, compared to state-of-the-art resource management approaches [18], [49], [46] also proposed as basic mechanisms by current IaaS systems [8], our model can yield a significant improvement in terms of efficiency, in the range 50-70%.

The remainder of the paper is organized as follows. Section II describes the reference system under study. In Section III we introduce our model based on the concept of GNE. In Section IV we prove the existence of at least a GNE and we provide two alternative solution methods. The experimental results discussed in Section V demonstrate the efficiency of our solutions. Other approaches proposed in the literature are discussed in Section VI. Conclusions are finally drawn in Section VII.

II. Problem statement and Assumptions

As stated in the previous Section, we consider SaaS providers using Cloud computing facilities according to the IaaS paradigm to offer multiple transactional Web services (WSs), each service representing a different application.

The hosted WSs can be heterogeneous with respect to resource demands, workload intensities and QoS requirements. The set of WS applications offered by the p-th SaaS provider are denoted by $\mathcal{A}_p$, while $\mathcal{P}$ will indicate the set of SaaSs.

An SLA contract, associated with each WS application, is established between the SaaS provider and its end-users. In particular, as in other approaches [11], [13], [43], for each WS application $k \in \mathcal{A}_p$, a linear utility function specifies the per request revenue (or penalty) $\mathcal{V}_k = \nu_k + m_k R_k$ incurred when the end-to-end response time R_k assumes a given value (see Figure 1). The slope of the utility function is $m_k = -\nu_k/\overline{R}_k < 0$ and $\overline{R}_k$ is the threshold that identifies the revenue/penalty region, that is, if $R_k > \overline{R}_k$ the SLA is violated and the SaaS incurs in penalties.

Linear utility functions are a flexible mechanism to rank different applications (e.g., assigning higher slopes to more important applications), and allow also to implement soft constraints on response times since the SaaS goal is to keep the infrastructure working in a profitability region, i.e., to provide an average response time lower than $\overline{R}_k$ looking for the trade-off between the SLA revenues and IaaS costs [11].

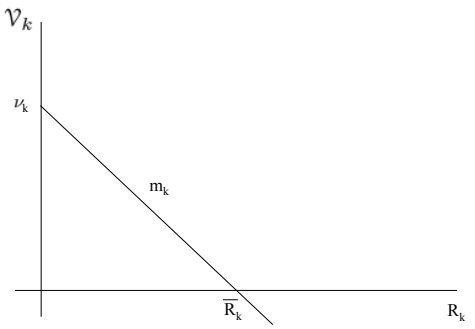

Fig. 1. Utility function regulating SaaS provider SLA contracts.

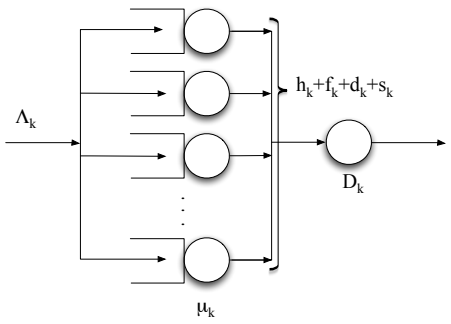

Fig. 2. System performance model.

Applications are hosted in virtual machines (VMs) which are dynamically instantiated by the IaaS provider. We make the simplifying assumption that each VM hosts a single WS application. Multiple VMs implementing the same WS application can also run in parallel.

IaaS providers usually charge software providers on an hourly basis [7]. Hence, the SaaS has to face the problem of determining every hour the optimal number of VMs for each WS class in order to maximize the net revenues, performing resource allocation on the basis of a prediction of future WS workloads [1], [13]. The SaaS needs also an estimate of the future performance of each VM in order to determine application average response time. In the following we model each WS class hosted in a VM as an M/G/1 queue in tandem with a delay center [14], as done in [34], [30], [2] and we assume, as common among Web service containers, 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, we suppose that the running VMs are homogeneous in terms of RAM and CPU capacity and the workload is evenly shared among multiple instances (see Figure 2). As discussed in [30] the delay center allows to model network delays and/or protocol delays introduced in establishing connections, etc.

For the IaaS provider we consider a model similar to Amazon EC2 [7]. The IaaS provider offers: (i) *flat* VMs, for which SaaS providers applies for a one-time payment (currently every one or three years) for each instance they want to reserve, (ii) *on demand* VMs, which allows SaaS to access computing capacity with no long-term commitments, and (iii) *on spot* VMs, for which SaaS providers bid and compete for unused infrastructure provider capacity. Furthermore, we assume that each SaaS provider owns, possibly, an enterprise infrastructure and implements an hybrid Cloud [12]. In this way the SaaS usually runs the applications in its own infrastructure whose capacity planning is performed considering the average workload and takes advantage

of the Cloud to manage workload peaks. For the sake of simplicity, we assume that the SaaS infrastructure is located in the same region of the IaaS site and hence the end users experience the same performance accessing the SaaS or the IaaS infrastructure, indifferently[1].

The VM instances are charged with the on spot cost σ_k for application k, which is set by the IaaS and fluctuates periodically depending on the IaaS provider time of the day energy costs and also on the supply and demand from SaaS for on spot VMs [7], [26]. Indeed, SaaS providers compete among them for the use of on spot VMs specifying the maximum cost σ_k^U for each application k they are willing to pay per instance hour. The on spot cost σ_k is fixed by the IaaS provider which can also decide to do not allocate any on spot instance to a SaaS. For example in the Amazon case on spot costs are available via the EC2 API [7] or by third party sites [41]. On spot costs fluctuate according to the time of the day and on the Cloud site region, and could be less or greater than the time unit cost φ_k for flat VMs hosting a given WS application k. On spot instances have been traditionally adopted to support batch computing intensive workloads [7]. However, since nowadays IaaS providers allow specifying autonomic policies which can dynamically allocate VM instances in few minutes as a reaction to failures (see, e.g., Amazon AWS Elastic Beanstalk [8]), we advocate the use of on spot instances also to support the execution of traditional WS applications. We assume that in house resources hosted at the SaaS location can be accessed at no cost. We denote with δ_k the cost for WS application k on demand instances; with the current pricing models, δ_k is strictly greater than φ_k and we assume $\delta_k > \sigma_k^U$ for all k. Indeed, since the IaaS provider can arbitrarily terminate on spot instances from a SaaS resource pool [7], no one is willing to pay for a less reliable resource a time unit cost higher than on demand instances which provide a higher availability level.

Finally, we will denote with h_p^U the overall number of VMs that can be hosted at the SaaS enterprise infrastructure (according to its capacity). The number of flat VMs each SaaS provider is guaranteed to have access to by applying to the one-time payment is denoted by f_p^U, while the number of on spot VMs available at the IaaS Cloud service center at a given time instant is denoted by s^U.

III. Generalized Nash game model

In this Section we formulate the resource provisioning problem for the Cloud computing system under study as a Generalized Nash Equilibrium Problem (GNEP). The goal of SaaS provider p is to determine every hour the number of in house h_k, flat f_k, on demand d_k, and on spot s_k VMs to

[1]This assumption does not change the nature of the problem under study and can be relaxed. See the next Section.

be devoted for the execution of all WS applications $k \in \mathcal{A}_p$, in order to maximize its profits and, at the same time, so as to to satisfy the prediction Λ_k for the arrival rate of the WS application k. Let us denote with μ_k the maximum service rate for the requests of application k. If the workload is evenly shared among the VMs, then the average response time for the execution of application k requests is given by:

$$E[R_k] = D_k + \frac{1}{\mu_k - \frac{\Lambda_k}{h_k+f_k+d_k+s_k}}, \tag{1}$$

where D_k denotes the queueing network delay (see Figure 2) and we further assume that the VMs are not saturated (i.e., the equilibrium conditions for the M/G/1 queues hold , $\mu_k\,(h_k + f_k + d_k + s_k) - \Lambda_k > 0$).

The average per time unit revenues for application k requests are given by $\mathcal{V}_k\,\Lambda_k = (\nu_k + m_k\,E[R_k])\,\Lambda_k$.

Considering the infrastructural costs to access flat, on demand, and on spot VM instances the goal of a SaaS provider is to maximize its profits given by:

$$\sum_{k\in\mathcal{A}_p}\left(\nu_k\,\Lambda_k + m_k\,\Lambda_k\,D_k + \frac{m_k\,\Lambda_k(h_k+f_k+d_k+s_k)}{\mu_k\,(h_k+f_k+d_k+s_k)-\Lambda_k} - \varphi\,f_k - \delta_k\,d_k - \sigma_k\,s_k\right)$$

With this setting in mind, the problem that the generic SaaS provider p has to periodically solve becomes:

$$\max\ \Theta_p = \sum_{k\in\mathcal{A}_p}\frac{m_k\,\Lambda_k(h_k+f_k+d_k+s_k)}{\mu_k\,(h_k+f_k+d_k+s_k)-\Lambda_k} - \sum_{k\in\mathcal{A}_p}\varphi\,f_k - \sum_{k\in\mathcal{A}_p}\delta_k\,d_k - \sum_{k\in\mathcal{A}_p}\sigma_k\,s_k$$

$$h_k,\ f_k,\ d_k,\ s_k \geq 0, \qquad \forall\ k \in \mathcal{A}_p, \tag{2}$$

$$h_k + f_k + d_k + s_k > \Lambda_k/\mu_k, \qquad \forall\ k \in \mathcal{A}_p, \tag{3}$$

$$s_k \leq \frac{\alpha_k}{1-\alpha_k}\,(h_k + f_k + d_k), \qquad \forall\ k \in \mathcal{A}_p, \tag{4}$$

$$\sum_{k\in\mathcal{A}_p} h_k \leq h_p^U, \tag{5}$$

$$\sum_{k\in\mathcal{A}_p} f_k \leq f_p^U, \tag{6}$$

$$\sum_{k\in\mathcal{A}} s_k \leq s^U, \tag{7}$$

where $\mathcal{A}$ denotes the set of indexes of all WS applications (i.e., $\mathcal{A} = \cup_p \mathcal{A}_p$, $\mathcal{A}_{p_1} \cap \mathcal{A}_{p_2} = \emptyset$ if $p_1 \neq p_2$). Note that the term $\sum_{k\in\mathcal{A}} \nu_k\,\Lambda_k + m_k\,\Lambda_k\,D_k$ can be dropped in the SaaS objective function since it is independent of the decision variables.

Constraint (3) guarantees that resources are not saturated. Constraint (4) is introduced for fault

tolerance reasons and guarantees that the on spot instances are at most a fraction $\alpha_k < 1$ than the overall resources allocated to application k. Indeed, since the on spot VMs are less reliable than flat and on demand, one can be conservative in their use since if only on spot VMs are adopted and they are terminated by the IaaS, then application k could become unavailable. Constraints (5) and (6) entail that the in house and flat VMs allocated to applications are less or equal to the ones available. Finally, constraint (7) guarantees than the on spot VMs allocated to competing SaaS providers are lower than the one available at the IaaS Cloud service center s^U.

We would like to remark that, in the formulation of the problem, we have not imposed variables h_k, f_k, d_k, s_k to be integer, as in reality they are. In fact, requiring variables to be integer makes the solution much more difficult. However, preliminary experimental results have shown that 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. We therefore decide to deal with continuous variables, actually considering a relaxation of the real problem.

On the other side, the IaaS provider's goal is to determine the time unit cost σ_k for on spot VM instances for all applications $k \in \mathcal{A}_p$ and every SaaS provider p, in order to maximize its total revenue:

$$\max \ \Theta_I = \sum_{k \in \mathcal{A}} (\varphi \, f_k + \delta_k \, d_k + \sigma_k \, s_k)$$
$$\sigma_k^L \leq \sigma_k \leq \sigma_k^U \qquad \forall \ k \in \mathcal{A}. \tag{8}$$

Note that the on spot instance cost lower bound σ_k^L is fixed according to the time of the day and includes the energy costs for running a single VM instance for one hour and the amortized cost of the hosting physical machine [26]. For the sake of clarity, the notation adopted in this paper is summarized in Table I.

If the maximum time unit cost of an application is lower than the minimum set by the IaaS, i.e. $\sigma_k^U < \sigma_k^L$, formally the SaaSs and IaaS problems have no solution. In that case we can set $s_k = 0$ and consider a simplified problem where the capacity allocation for application k is limited to determine the number of in house, flat, and on demand instances. Hence, in the following we will always assume that $\sigma_k^L \leq \sigma_k^U$ for all k. Note that, if the on spot instances are terminated by the IaaS provider, then the SaaS can dynamically start the execution of an equal number of on demand instances.

Decision Variables

h_k	Number of *in house* VMs used for application k	f_k	Number of *flat* VMs used for application k
d_k	Number of *on demand* VMs used for application k	s_k	Number of *on spot* VMs used for application k
σ_k	Time unit cost for *on spot* VMs used for application k		

System Parameters

$\mathcal{P}$	Set of SaaS providers
$\mathcal{A}_p$	Set of applications of the p SaaS provider
$\mathcal{A}$	Set of applications of all the SaaS providers
h_p^U	Maximum number of in house VMs that can be hosted at the SaaS provider p facility
f_p^U	Maximum number of flat computational resources available for provider p
s^U	Maximum number of on spot computational resources IaaS can provide to all the SaaS providers
Λ_k	Prediction of the arrival rate for application k
μ_k	Maximum service rate for executing class k application
D_k	Queueing delay for executing class k application
m_k	Application k utility function slope
α_k	Maximum fraction of resources allocated as on spot VMs for application k
φ_k	Time unit cost for *flat* VMs used for application k
δ_k	Time unit cost for *on demand* VMs used for application k
σ_k^L	Minimum time unit cost for *on spot* VMs used for application k, set by the IaaS provider
σ_k^U	Maximum time unit cost for *on spot* VMs used for application k, set by the SaaS provider

TABLE I

PARAMETERS AND DECISION VARIABLES.

In this framework, SaaS providers and the IaaS provider are making decisions at the same time: The decisions of a SaaS depend on those of the others SaaS and the IaaS, while the IaaS objective function depends on SaaS decisions. In this setting, we can not analyze decision in isolation, but we must ask what a SaaS would do, taking into account the decision of the IaaS and other SaaSs. To capture the behavior of SaaSs and IaaS in this conflicting situation (game) in which what a SaaS or the IaaS (the players of the game) does directly affects what others do, we consider the Generalized Nash game [20], [22], which is broadly used in Game Theory and other fields. We remind the reader that the GNEP differs from the classical Nash Equilibrium Problem since, not only the objective functions of each player (called *payoff functions*) depend upon the strategies chosen by all the other players, but also each player's strategy set may depend on the rival players' strategies. In our setting the constraint of each problem involving other player's variables (joint constraint) comes from (7).

Following the Nash equilibrium concept, SaaS and IaaS providers adopt a strategy such that none of them can improve its revenue by changing its strategy unilaterally (i.e., while the other players keep their strategies unchanged). The service provisioning problem results therefore in a GNEP where the players are the SaaS providers and the IaaS provider, the strategy variables of SaaS provider p are h_k, f_k, d_k, and s_k, for $k \in \mathcal{A}_p$, while the strategy variables of the IaaS are

the costs for *on spot* VMs, σ_k, for all $k \in \mathcal{A}$.

Another important feature of the derived GNEP is that it satisfies the *Convexity Assumption*: The payoff functions of both SaaS providers and IaaS, are concave in its own variables (Θ_p is concave being the sum of linear and concave function[2], and Θ_I is linear) and the set of strategies are convex (only linear constraints are introduced). Moreover, even if the decision of a SaaS depends on the decisions of the other SaaSs and the IaaS, the constraint of each problem involving other player's variables (coming from (7) in each SaaS problem), is the same for all players: We refer to this special class of GNEP as *jointly convex* GNEP [22].

IV. Solution methods

In this Section we deal with existence, properties, and algorithms for the solutions of the generalized Nash game previously described. In particular, in the next Section we prove the existence of at least one social equilibrium and we determine the conditions which guarantee that a GNE is also a social equilibrium. In Section IV-B we provide two different methods for identifying GNE.

A. Social and Nash equilibria for the Service Provisioning Game

To simplify the discussion we introduce the following notations. Let $x^p = (h_k, f_k, d_k, s_k)_{k \in \mathcal{A}_p}$ denotes the strategies vector of SaaS provider p, x^{-p} the vector formed by the strategies of all SaaS providers different from p, and $x = (x^p)_{p \in \mathcal{P}}$. Moreover we indicate by $X_p(x^{-p})$ the set of strategies for provider p and X the set of vectors x satisfying constraints (2)–(7).

First, we note that the IaaS's behaviour is simple. In fact, if a SaaS provider decides to use on spot VMs for application k, that is $s_k > 0$, then the best response of the IaaS is to play $\sigma_k = \sigma_k^U$; otherwise if $s_k = 0$, the value of the IaaS payoff is constant and independent of his choice σ_k that can be any value in the interval $[\sigma_k^L, \sigma_k^U]$. Therefore, the strategy $\sigma_k = \sigma_k^U$ for all $k \in \mathcal{A}$ is a dominant strategy for the IaaS provider. Hereafter we suppose, without loss of generality, that IaaS provider plays this strategy. Assuming that IaaS chooses his dominant strategy, the payoff function Θ_p of each SaaS provider p only depends on his own strategy x^p. This is a potential

[2]The concavity of each summand $\frac{m_k \Lambda_k (h_k+f_k+d_k+s_k)}{\mu_k (h_k+f_k+d_k+s_k)-\Lambda_k}$ follows by direct evaluation of the Hessian. The eigenvalues are zero and $\frac{8 m_k \mu_k \Lambda_k^2}{[\mu_k (h_k+f_k+d_k+s_k)-\Lambda_k]^3}$, which is negative since $m_k < 0$ and $h_k + f_k + d_k + s_k > \Lambda_k/\mu_k$. Note that, if we assume that the SaaS and IaaS are located in different regions and hence the end user requests redirected from the SaaS to the IaaS site experience a significant delay, then our model can be extended by introducing two delays centres D_k^1 for the requests served in house, and D_k^2 for the requests served at the IaaS infrastructure. In that case, Θ_p introduces additional convex terms like $m_k \Lambda_k \left(D_1^k h_k/(h_k + f_k + d_k + s_k) + D_2^k (f_k + d_k + s_k)/(h_k + f_k + d_k + s_k)\right)$ without altering the nature of the problem under study.

game [35], where the sum of all payoff functions:

$$\Pi(x) = \sum_{p \in \mathcal{P}} \Theta_p(x^p)$$

is the potential function. This means that the GNE of our game coincide with those of a game where all the payoff functions are equal to the potential Π.

The maximizers of Π on the set X are called *social equilibria* of the game. It is clear from the above definitions that each social equilibrium is a special GNE, indeed no one player can improve its payoff by unilaterally or *collectively* deviating its strategy. In other words, social equilibria represent the GNE which are optimal from a social point of view. We now show the existence of social equilibria for our problem.

Theorem 1: There exists at least one social equilibrium of the game.

Proof: We exploit the well-known Weierstrass theorem, which guarantees that a continuous function admits a maximizer on a closed and bounded set. The potential function is defined as:

$$\Pi(x) = \sum_{k \in \mathcal{A}} \left[\frac{m_k \, \Lambda_k (h_k + f_k + d_k + s_k)}{\mu_k \, (h_k + f_k + d_k + s_k) - \Lambda_k} - \varphi \, f_k - \delta_k \, d_k - \sigma_k^U \, s_k \right].$$

It is continuous, but the set X is neither closed nor bounded. However, we remark that if $h_k + f_k + d_k + s_k \to \Lambda_k / \mu_k$ for some $k \in \mathcal{A}_p$, then $\Pi(x) \to -\infty$. Moreover, if $d_k \to +\infty$ then $\Pi(x) \to -\infty$ as well. Therefore there exists a suitable value v such that the superlevel set $\{x \in X : \ \Pi(x) \geq v\}$ is closed and bounded. Since the maximizers of Π on X coincide with the maximizers of Π on this superlevel set, the existence of a social equilibrium follows from the Weierstrass theorem. ■

We remark that there can be multiple social equilibria (the potential function is concave but not strictly concave); however the value of all social equilibria is obviously unique.

A GNE is not necessarily a social equilibrium. However, it is possible to identify which Nash equilibria are social equilibria by means of a property on Karush-Kuhn-Tucker (KKT) multipliers. We rewrite the problem of SaaS provider p as:

$$\begin{cases} \max_{x^p} \ \Theta_p(x^p) \\ g_p(x^p) \leq 0 \\ z(x) \leq 0, \end{cases} \tag{9}$$

where $g_p(x^p) \leq 0$ summarizes individual constraints (2)–(6) and $z(x) \leq 0$ is the joint constraint (7). We denote by β_p and γ_p the optimal KKT multipliers associated to $g_p(x^p) \leq 0$ and $z(x) \leq 0$, respectively.

Theorem 2: If $\overline{x} = (\overline{x}^1, \ldots, \overline{x}^{|\mathcal{P}|})$ is a GNE of the game and the KKT multipliers $(\overline{\beta}, \overline{\gamma})$ associated to $\overline{x}$ satisfy the relation $\overline{\gamma}_1 = \cdots = \overline{\gamma}_{|\mathcal{P}|}$, then $\overline{x}$ is a social equilibrium.

Proof: Since $\overline{x}$ is a Nash equilibria, $\overline{x}^p$ is an optimal solution of problem (9) $\forall p \in \mathcal{P}$. Therefore the relative KKT multipliers $\overline{\beta}_p$ and $\overline{\gamma}_p$ satisfy the following system:

$$\begin{cases} \nabla\Theta_p(\overline{x}^p) + \overline{\beta}_p^T \nabla g_p(\overline{x}^p) + \overline{\gamma}_p \nabla_{x^p} z(\overline{x}) = 0 \\ \overline{\beta}_p^T g_p(\overline{x}^p) = 0 \\ \overline{\gamma}_p \, z(\overline{x}) = 0 \\ \overline{\beta}_p \leq 0, \;\; g_p(\overline{x}^p) \leq 0 \\ \overline{\gamma}_p \leq 0, \;\; z(\overline{x}) \leq 0. \end{cases}$$

From the definition of the potential function we have that $\nabla\Theta_p(\overline{x}^p) = \nabla_{x^p}\Pi(\overline{x})$. Taking into account the assumption on multipliers $\overline{\gamma}_1, \ldots, \overline{\gamma}_{|\mathcal{P}|}$ we obtain the following system:

$$\begin{cases} \nabla_{x^p}\Pi(\overline{x}) + \overline{\beta}_p^T \nabla g_p(\overline{x}^p) + \overline{\gamma}_1 \nabla_{x^p} z(\overline{x}) = 0 \quad \forall \; p \in \mathcal{P} \\ \overline{\beta}_p^T g_p(\overline{x}^p) = 0 \quad \forall \; p \in \mathcal{P} \\ \overline{\gamma}_1 \, z(\overline{x}) = 0 \\ \overline{\beta}_p \leq 0, \;\; g_p(\overline{x}^p) \leq 0 \quad \forall \; p \in \mathcal{P} \\ \overline{\gamma}_1 \leq 0, \;\; z(\overline{x}) \leq 0, \end{cases}$$

that is $(\overline{x}, \overline{\beta}, \overline{\gamma}_1)$ solves the KKT system associated to the problem $\max_{x \in X} \Pi(x)$. Since Π is a concave function and X is defined by linear constraints, we obtain that $\overline{x}$ is a maximizer of Π on X, that is $\overline{x}$ is a social equilibrium. ■

B. Algorithms for identifying Generalized Nash Equilibria

We now present a first algorithm (see Figure 3), based on the best-reply dynamics, for finding GNE of the game. Starting from a feasible vector, each player solves in turn his own optimization problem (steps 0, 1). After that, if joint constraint (7) has become active or if all the players have multiplier γ_p equal to zero, then the algorithm has found a GNE (step 2). Otherwise, we have not a GNE because some on spot VMs are available (constraint (7) is not active), but there are also players (with $\gamma_p < 0$) wishing more on spot resources than those computed by the algorithm so far. Thus, the players with multiplier $\gamma_p < 0$ solve for the second time in turn his own optimization problem and the algorithms stops with a GNE (step 3). The condition leading to step 3 can occur when, in step 1, first some players saturate the use of on spot resources, then other players find

0. (Inizialization) Set $\sigma_k = \sigma_k^U$ for all $k \in \mathcal{A}$ and select $\widetilde{x} \in X$.
1. (First round of best-reply dynamics)
for all $p \in \mathcal{P}$ **do**
Find an optimal solution $\overline{x}^p$ of $\begin{cases} \max_{x^p} \ \Theta_p(x^p) \\ g_p(x^p) \leq 0 \\ z(x^p, \widetilde{x}^{-p}) \leq 0 \end{cases}$
Let $\overline{\gamma}_p$ be the optimal multiplier relative to constraint $z(x) \leq 0$
Set $\widetilde{x}^p = \overline{x}^p$
end
2. (Stopping criterion)
if $\overline{\gamma}_p = 0$ for all $p \in \mathcal{P}$ **or** $z(\widetilde{x}) = 0$ **then** STOP
3. (Possible second round of best-reply dynamics)
for all $p \in \mathcal{P}$ **do**
if $\overline{\gamma}_p < 0$ **then**
Find an optimal solution $\overline{x}^p$ of $\begin{cases} \max_{x^p} \ \Theta_p(x^p) \\ g_p(x^p) \leq 0 \\ z(x^p, \widetilde{x}^{-p}) \leq 0 \end{cases}$
Set $\widetilde{x}^p = \overline{x}^p$
if $z(\widetilde{x}) = 0$ **then** STOP
end
end

Fig. 3. Algorithm 1 for finding Generalized Nash Equilibria.

an optimal value of on spot resources lower than the initial one established at step 0 releasing common resources.

Theorem 3: Algorithm 1 finds a Generalized Nash Equilibrium.

Proof: At the end of step 1 each player p has computed the best reply $\overline{x}^p$ to strategy $\widetilde{x}^{-p}$ of the other players. Therefore the following KKT system holds $\forall p \in \mathcal{P}$:

$$\begin{cases} \nabla \Theta_p(\overline{x}^p) + \overline{\beta}_p^T \nabla g_p(\overline{x}^p) + \overline{\gamma}_p \nabla_{x_p} z(\overline{x}^p, \widetilde{x}^{-p}) = 0 \\ \overline{\beta}_p^T \, g_p(\overline{x}^p) = 0 \\ \overline{\gamma}_p \, z(\overline{x}^p, \widetilde{x}^{-p}) = 0 \\ \overline{\beta}_p \leq 0, \ \ g_p(\overline{x}^p) \leq 0 \\ \overline{\gamma}_p \leq 0, \ \ z(\overline{x}^p, \widetilde{x}^{-p}) \leq 0. \end{cases}$$

From this system we can deduce that if $\overline{\gamma}_p = 0$ then $\overline{x}^p$ is the best reply of player p also to every feasible strategy x^{-p}, i.e. such that $z(\overline{x}^p, x^{-p}) \leq 0$. Conversely, since $\nabla_{x_p} z$ is a constant vector and independent from x^{-p}, we obtain that if $\overline{\gamma}_p < 0$, then $\overline{x}^p$ is the best reply of player p to every strategy x^{-p} of the other players provided that $z(\overline{x}^p, x^{-p}) = 0$.

The previous discussion proves the correctness of the stopping criterion at step 2. In fact, if at the end of step 1 we have $\overline{\gamma}_p = 0$ for all p or $z(\widetilde{x}) = 0$, then the strategy $\overline{x}^p$ of player p is the best reply to strategies $\overline{x}^{-p}$ of other players, i.e. $\overline{x}$ is a GNE.

When there are some multipliers $\overline{\gamma}_p < 0$ and $z(\widetilde{x}) < 0$, then the players with $\overline{\gamma}_p < 0$ have strategy $\overline{x}^p$ which is not the best reply to the other players' strategies. Therefore, at step 3 we recompute the best reply strategy for these players. If during the execution of step 3 we obtain $z(\widetilde{x}) = 0$, then we can prove as before that a GNE has been found. Otherwise, if $z(\widetilde{x}) < 0$ after each new best reply calculation, then all the players have multipliers $\gamma_p = 0$ and thus we obtain also in this case a GNE. ■

We remark that Algorithm 1 finds a GNE after resolving at least $|\mathcal{P}|$ and no more than $2\,|\mathcal{P}| - 1$ optimization problems.

We now present another algorithm (see Figure 4) determining a GNE which is simpler than Algorithm 1. At step 1 each SaaS provider p finds the optimal solution $\widetilde{x}^p$ of his relaxed problem where the joint constraint (7) is removed. If the solution $\widetilde{x} = (\widetilde{h}, \widetilde{f}, \widetilde{d}, \widetilde{s})$ satisfies constraint (7), then it is a social equilibrium. Otherwise, the on spot VMs are shared proportionally among SaaS providers according to the relaxed solutions, i.e. we set a new upper bound $s_p^U = (s^U \sum_{k \in \mathcal{A}_p} \widetilde{s}_k) / \sum_{k \in \mathcal{A}} \widetilde{s}_k$ for the on spot resources of each player p. Finally, each SaaS provider solves his problem with this new bound and a GNE is found.

0. (Inizialization) Set $\sigma_k = \sigma_k^U$ for all $k \in \mathcal{A}$.
1. (Solution of relaxed subproblems)
   for all $p \in \mathcal{P}$ **do**
   Find an optimal solution $\widetilde{x}^p$ of $\begin{cases} \max\ \Theta_p(x^p) \\ g_p(x^p) \leq 0 \end{cases}$
   end
2. (Stopping criterion)
   if $z(\widetilde{x}) \leq 0$ **then** STOP
   else set $s_p^U = \dfrac{\sum_{k \in \mathcal{A}_p} \widetilde{s}_k}{\sum_{k \in \mathcal{A}} \widetilde{s}_k}\, s^U$ for all $p \in \mathcal{P}$
3. (Solution of new subproblems)
   for all $p \in \mathcal{P}$ **do**
   Find an optimal solution $\overline{x}^p$ of $\begin{cases} \max_{x^p}\ \Theta_p(x^p) \\ g_p(x^p) \leq 0 \\ \sum_{k \in \mathcal{A}_p} s_k \leq s_p^U \end{cases}$
   end

Fig. 4. Algorithm 2 for finding Generalized Nash Equilibria.

Theorem 4: Algorithm 2 finds a Generalized Nash Equilibrium.

Proof: At step 1 each provider p finds individually the optimal solution $\widetilde{x}^p$ of the relaxed problem, thus there are optimal KKT multipliers $\widetilde{\beta}_p$ such that the following system holds $\forall p \in \mathcal{P}$:

$$\begin{cases} \nabla\Theta_p(\widetilde{x}^p) + \widetilde{\beta}_p^T \nabla g_p(\widetilde{x}^p) = 0, \\ \widetilde{\beta}_p^T g_p(\widetilde{x}^p) = 0, \\ \widetilde{\beta}_p \leq 0,\ g_p(\widetilde{x}^p) \leq 0. \end{cases}$$

If $\widetilde{x}$ also satisfies the joint constraint, then $(\widetilde{x}, \widetilde{\beta}, \widetilde{\gamma})$, with $\widetilde{\gamma} = 0$, solves the KKT system associated to the problem $\max_{x \in X} \Pi(x)$. Since Π is concave and X is defined by linear constraints, it follows that $\widetilde{x}$ is a maximizer of Π on X, i.e. a social equilibrium.

Otherwise, if the joint constraint is not satisfied at $\widetilde{x}$, i.e. $\sum_{k \in \mathcal{A}} \widetilde{s}_k > s^U$, then the on spot resources are proportionally rescaled for each player p to obtain a new individual upper bound s_p^U lower than the optimal amount of on spot VMs $\sum_{k \in \mathcal{A}_p} \widetilde{s}_k$ computed at step 1. Therefore, each optimal solution $\overline{x}^p$ at step 3 makes active the constraint $\sum_{k \in \mathcal{A}_p} s_k \leq s_p^U$ and the associated KKT multiplier is negative. Thus there are KKT multipliers $(\overline{\beta}_p, \overline{\gamma}_p)$, with $\overline{\gamma}_p < 0$, such that the following system holds $\forall p \in \mathcal{P}$:

$$\begin{cases} \nabla\Theta_p(\overline{x}^p) + \overline{\beta}_p^T \nabla g_p(\overline{x}^p) + \overline{\gamma}_p \nabla z_p(\overline{x}^p) = 0, \\ \overline{\beta}_p^T g_p(\overline{x}^p) = 0, \\ \overline{\beta}_p \leq 0,\ g_p(\overline{x}^p) \leq 0, \\ \overline{\gamma}_p < 0,\ z_p(\overline{x}^p) = 0, \end{cases}$$

where $z_p(x_p) \leq 0$ denotes the constraint $\sum_{k \in \mathcal{A}_p} s_k \leq s_p^U$. Since $z_p(\overline{x}^p) = 0$, i.e. $\sum_{k \in \mathcal{A}_p} \overline{s}_k = s_p^U$, we obtain:

$$\sum_{k \in \mathcal{A}} \overline{s}_k = \sum_{p \in \mathcal{P}} \sum_{k \in \mathcal{A}_p} \overline{s}_k = \sum_{p \in \mathcal{P}} s_p^U = s^U,$$

hence the joint constraint satisfies $z(\overline{x}) = 0$. Moreover, we note that $\nabla z_p(x_p) = \nabla_{x_p} z(x)$, therefore we obtain the following system $\forall p \in \mathcal{P}$:

$$\begin{cases} \nabla\Theta_p(\overline{x}^p) + \overline{\beta}_p^T \nabla g_p(\overline{x}^p) + \overline{\gamma}_p \nabla_{x_p} z(\overline{x}) = 0, \\ \overline{\beta}_p^T g_p(\overline{x}^p) = 0, \\ \overline{\beta}_p \leq 0,\ g_p(\overline{x}^p) \leq 0, \\ \overline{\gamma}_p < 0,\ z(\overline{x}) = 0. \end{cases}$$

This means that $\overline{x}^p$ is the optimal solution of the problem:

$$\begin{cases} \max_{x^p} \ \Theta_p(x^p) \\ g_p(x^p) \leq 0 \\ z(x^p, \overline{x}^{-p}) \leq 0 \end{cases}$$

that is $\overline{x}^p$ is the best reply of player p to the strategies $\overline{x}^{-p}$ of the other players. i.e. $\overline{x}$ is a GNE.

■

Note that, Algorithm 2 finds a GNE after resolving exactly $|\mathcal{P}|$ optimization problems, when the joint constraint (7) is satisfied at $\widetilde{x}$ and $2\,|\mathcal{P}|$ otherwise.

As a final remark it is worth noticing that Algorithm 1 has to be executed at the IaaS provider site, since the best reply problems at step 1 and step 3 of Figure 3 have to be solved sequentially. Alternatively, the IaaS provider has to coordinate the SaaS providers setting an initial feasible value (step 0) for SaaS provider p on spot resources $(\widetilde{s}_k)_{k\in\mathcal{A}_p}$ and setting also an upper bound $s^U - \sum_{k\in\mathcal{A}_p} \widetilde{s}_k$ for the remaining SaaS providers.

Vice versa, Algorithm 2 can be implemented in a fully distributed manner: The SaaS providers initially send to the IaaS their bid σ_k^U and the required value for the on spot resources, i.e., $\widetilde{s}_k$ obtained solving the relaxed sub-problem at step 1. Then, the IaaS provider sends back to individual SaaSs the effective on spot cost and the number of on spot VMs available s_p^U. In this latter case, each SaaS provider solves independently its individual sub-problem at step 3. Finally each SaaS starts on spot, on demand, and flat VMs according to the determined solution. As a final remark note that Algorithm 2 does not require to share the information on the SLA contracts and performance parameters (i.e., m_k, μ_k, etc.) among SaaSs and IaaS.

V. Numerical Analysis

The resource management algorithms proposed have been evaluated for a variety of system and workload configurations. Section V-A is devoted to quantitatively analyse the efficiency of the equilibria achieved by our approach, while the scalability is discussed in Section V-B. Section V-C illustrates the equilibria properties for a medium size system by varying application performance parameters. Finally, Section V-D shows the results of the application of our resource allocation techniques in a real prototype environment deployed on Amazon EC2.

A. Equilibria Efficiency

To evaluate the efficiency of our algorithms we have considered a very large set of randomly generated instances. The number of SaaS providers has been varied between 10 and 100, the

s^U	[100,1000]	m_k	[-10,-1]	Λ_k	[1,100] req/s	μ_k	[1,10] req/s
φ_k	[0.03,0.24]$	δ_k	[0.08,1.24]$	σ_k^L	[0.02,0.08]$	σ_k^U	[0.09,0.30]$

TABLE II

PERFORMANCE PARAMETERS AND TIME UNIT COST RANGES.

number of applications (evenly shared among SaaSs) between 100 and 1,000.

The performance parameters of Web applications and infrastructural resources costs have been randomly generated uniformly in the ranges reported in Table II as in other literature approaches [2], [11], [32] and according to commercial fees applied by IaaS Cloud providers [7]. The number of in house VMs h_p^U has been obtained by varying the capacity ratio ρ of the system, that is we have:

$$h_p^U = \frac{\rho}{\overline{U}} \sum_{k \in \mathcal{A}_p} \frac{\Lambda_k}{\mu_k}.$$

In other terms for each provider p, h_p^U has been obtained as a fraction ρ of the number of VMs required to serve the overall incoming workload with maximum utilization $\overline{U}$. As in other approaches considered in the literature [18], [49], [46] we set $\overline{U} = 0.6$, and ρ has been set equal to 0, i.e. a purely Cloud based system is considered, 0.5 i.e. the in house resources can manage around 50% of the peak workload, and 0.1 i.e. the in house resources can manage only 10% of the peak workload. Note that, this latter case is representative of scenarios where SaaS providers are characterized by highly variable workloads and want to effectively exploit the on demand nature of the Cloud.

The lower is the value of ρ and the higher is the use of Cloud resources for the SaaS providers. The analyses have been performed also varying the α_k parameters: the values 0.1, 0.5, and 0.8 have been considered. For the equilibria analysis the value of s^U has been randomly generated guaranteeing that the joint constraint (7) is active at each GNE found (this has been obtained by solving, for each player, problem (9) without the joint constraint, and setting s^U equal to 80% of the total on spot resources actually used).

We denote with $\overline{x}$ the social equilibrium and with $\widetilde{x}$ the equilibrium found by Algorithms 1 and 2. The efficiency has been measured in terms of the *Price of Anarchy* (PoA) and of the *Individual Worst Case* (IWC) evaluated as:

$$PoA = \frac{\Pi(\overline{x})}{\Pi(\widetilde{x})}, \qquad IWC = \max_{p \in \mathcal{P}} \frac{\Theta_p(\overline{x})}{\Theta_p(\widetilde{x})}.$$

Both PoA and IWC are a measure of the inefficiency due to SaaSs selfish behaviour with

respect to the scenario where the *social optimum* is pursued. In particular, the IWC is a measure of the gap between the social optimum and a GNE achieved in the worst case by a SaaS provider. Furthermore, we compared our solutions with another heuristic based on the utilization thresholds principle proposed in other literature approaches [18], [49], [46]. In particular for each WS application:

- The number of in house VMs is evenly shared among applications;
- The number of on spot VMs is assigned by the IaaS proportionally to the bid;
- The number of flat and on demand VMs is determined such that average utilization of all VMs is equal to $\overline{U}$. First flat VMs are used since they are cheaper than on demand ones.

Note that, if a particular WS application is under light load and hence a utilization lower than $\overline{U}$ can be obtained by adopting only in house and, possibly, on spot VMs, then a lower number of resources is used such that the utilization of running instances is equal to $\overline{U}$. In that case, the exceeding in house and on spot resources are evenly shared among the remaining WS applications. Threshold based approaches are advocated also by IaaS providers. For example, Amazon AWS Elastic Beanstalk [8] provides basic mechanism to trigger the start up or termination of VM instances according to the threshold values which can be specified by SaaS providers accessing the Amazon EC2 API.

Results are reported in Tables III-V. The figures are the means computed on ten different runs. In particular, Table III includes also the number of second round optimizations performed by Algorithm 1 at step 3. This value is very low and in practice the overall number of optimization problems solved by Algorithm 1 is very close to $|\mathcal{P}|$. From the efficiency point of view, Algorithms 1 and 2 provides similar results: The PoA is lower than 1.01 (i.e., on average the percentage difference of the sum of the payoff functions with respect to the social optimum is lower than 1%), while the IWC is lower than 1.12 (i.e., in the worst case the revenue of a SaaS provider in the social equilibrium is 12% greater than in the GNE). As expected the PoA and IWC are higher when the SaaS providers significantly use the Cloud resources (e.g., because they do not own in house private infrastructure $\rho = 0$, or they are not conservative in the use of on spot resources, i.e., α_k is high). Vice versa, the heuristic solution is less far efficient than Algorithms 1 and 2 since the average value for PoA and IWC are around 1.42 and 1.62, respectively. Furthermore, these values are almost independent of the number of SaaS and WS applications but are very sensitive to ρ

	$\alpha = 0.1, \rho = 0.5$			$\alpha = 0.5, \rho = 0.5$			$\alpha = 0.1, \rho = 0.1$			$\alpha = 0.5, \rho = 0.1$		
$(\lvert\mathcal{P}\rvert, \lvert\mathcal{A}\rvert)$	PoA	IWC	# Reopt	PoA	IWC	# Reopt	PoA	IWC	# Reopt	PoA	IWC	# Reopt
(10,100)	1.0005	1.0060	10.3	1.0063	1.0302	12.0	1.0004	1.0051	9.4	1.0052	1.0292	10.6
(20,200)	1.0004	1.0058	10.8	1.0041	1.0332	11.3	1.0002	1.0049	7.7	1.0036	1.0318	10.3
(30,300)	1.0003	1.0063	12.7	1.0038	1.0348	10.2	1.0003	1.006	14.4	1.0034	1.0333	10.0
(40,400)	1.0004	1.0076	12.8	1.0039	1.0413	8.3	1.0003	1.0062	14.5	1.0034	1.0389	8.2
(50,500)	1.0004	1.0072	10.2	1.0042	1.0444	6.9	1.0003	1.0070	8.5	1.0036	1.0431	6.0
(60,600)	1.0004	1.0099	16.2	1.0051	1.0582	9.4	1.0003	1.0086	15.1	1.0037	1.0566	13.0
(70,700)	1.0004	1.0091	13.0	1.0036	1.0551	7.5	1.0003	1.0072	13.4	1.0027	1.0507	9.3
(80,800)	1.0002	1.008	25.0	1.0021	1.0494	7.9	1.0002	1.0070	22.4	1.0016	1.0469	7.9
(90.,900)	1.0003	1.0085	13.5	1.0027	1.0602	5.6	1.0002	1.0082	13.3	1.0019	1.0564	5.9
(100,1000)	1.0005	1.0097	9.2	1.0050	1.0744	0.6	1.0003	1.0091	13.8	1.0028	1.0681	0.5

	$\alpha = 0.1, \rho = 0$			$\alpha = 0.5, \rho = 0$			$\alpha = 0.8, \rho = 0$		
$(\lvert\mathcal{P}\rvert, \lvert\mathcal{A}\rvert)$	PoA	IWC	# Reopt	PoA	IWC	# Reopt	PoA	IWC	# Reopt
(10,100)	1.0002	1.0049	13.6	1.0006	1.0239	13.1	1.0054	1.0323	13.9
(20,200)	1.0002	1.0051	11.4	1.0016	1.0303	8.4	1.0037	1.0476	9.3
(30,300)	1.0002	1.0057	19.1	1.001	1.0302	16.6	1.0045	1.0440	16.1
(40,400)	1.0003	1.0076	13.2	1,0017	1.0471	11.3	1.0060	1,0795	8.2
(50,500)	1.0002	1.0067	12.6	1.0010	1.0371	10.8	1.0075	1.0872	6.7
(60,600)	1.0003	1.0085	20.4	1.0023	1.0542	12.8	1.0061	1.1065	3.4
(70,700)	1.0002	1.0074	19.7	1.0028	1.0499	3.3	1.0033	1.0964	1.2
(80,800)	1.0003	1.007	18.0	1.0039	1.0482	6.1	1.0028	1.0896	1.8
(90,900)	1.0002	1.0078	12.1	1.0043	1.0502	3.5	1.0023	1.097	0.0
(100,1000)	1.0003	1.0105	10.7	1.0044	1.0617	1.3	1.004	1.113	0.6

TABLE III

ALGORITHM 1 EFFICIENCY.

	$\alpha=0.1,\rho=0.5$		$\alpha=0.5,\rho=0.5$		$\alpha=0.1,\rho=0.1$		$\alpha=0.5,\rho=0.1$		$\alpha=0.1,\rho=0$		$\alpha=0.5,\rho=0$		$\alpha=0.8,\rho=0$	
$(\lvert\mathcal{P}\rvert,\lvert\mathcal{A}\rvert)$	PoA	IWC	PoA	IWC	PoA	IWC	PoA	IWC	PoA	IWC	PoA	IWC	PoA	IWC
(10,100)	1.0005	1.0020	1.0014	1.005	1.0004	1.0018	1.0002	1.0022	1.0003	1.0019	1.0006	1.0033	1.0009	1.0055
(20,200)	1.0004	1.0018	1.0019	1.0123	1.0003	1.0016	1.0006	1.0022	1.0003	1.0016	1.0016	1.0102	1.0025	1.0225
(30,300)	1.0003	1.0017	1.0016	1.0117	1.0003	1.0016	1.0003	1.0081	1.0003	1.0016	1.0010	1.0087	1.0029	1.0214
(40,400)	1.0004	1.0021	1.0022	1.0180	1.0003	1.0016	1.0001	1.0091	1.0003	1.0019	1.0017	1.0180	1.0047	1.0600
(50,500)	1.0004	1.0020	1.0029	1.0205	1.0003	1.0019	1.0007	1.0064	1.0003	1.0023	1.001	1.0127	1.0056	1.0584
(60,600)	1.0004	1.0032	1.0031	1.0333	1.0003	1.0029	1.0008	1.0078	1.0003	1.0023	1.0023	1.0291	1.0062	1.0836
(70,700)	1.0005	1.0030	1.0045	1.0411	1.0003	1.0026	1.0003	1.0014	1.0003	1.0032	1.0028	1.0399	1.0052	1.0870
(80,800)	1.0004	1.0028	1.0046	1.0412	1.0004	1.0025	1.0052	1.0229	1.0004	1.002	1.0039	1.0348	1.0065	1.0792
(90,900)	1.0004	1.0032	1.0051	1.0473	1.0004	1.0029	1.0057	1.0332	1.0004	1.0031	1.0043	1.0419	1.0061	1.0918
(100,1000)	1.0006	1.0035	1.0069	1.0591	1.0005	1.0033	1.0056	1.0395	1.0005	1.0041	1.0044	1.0454	1.0060	1.1043

TABLE IV

ALGORITHM 2 EFFICIENCY.

	$\alpha = 0.1, \rho = 0.5$		$\alpha = 0.5, \rho = 0.5$		$\alpha = 0.1, \rho = 0.1$		$\alpha = 0.5, \rho = 0.1$		$\alpha = 0.1, \rho = 0$		$\alpha = 0.5, \rho = 0$		$\alpha = 0.8, \rho = 0$		
$(\lvert\mathcal{P}\rvert, \lvert\mathcal{A}\rvert)$	PoA	IWC	PoA	IWC	PoA	IWC	PoA	IWC	PoA	IWC	PoA	IWC	PoA	IWC	
(10,100)	1.4293	1.5604	1.2914	1.5337	1.4779	1.5868	1.3833	1.6038	1.4819	1.5879	1.3155	1.5262	1.2259	1.4310	1.3720
(20,200)	1.4696	1.6282	1.3210	1.5900	1.5427	1.6521	1.3670	1.5997	1.5615	1.6582	1.4155	1.6431	1.3665	1.6272	1.4348
(30,300)	1.4984	1.6590	1.3304	1.5774	1.5734	1.6912	1.4139	1.629	1.5874	1.6895	1.4304	1.6695	1.3813	1.6429	1.4593
(40,400)	1.4582	1.6298	1.3201	1.5967	1.3739	1.4815	1.3804	1.6026	1.5312	1.6441	1.4053	1.6406	1.3736	1.6541	1.4061
(50,500)	1.4835	1.6472	1.3465	1.6417	1.5479	1.6694	1.4784	1.7146	1.5475	1.6723	1.4020	1.6643	1.3889	1.7076	1.4563
(60,600)	1.4062	1.5781	1.2803	1.5974	1.4584	1.5914	1.3422	1.608	1.4753	1.5942	1.3586	1.6146	1.3446	1.6791	1.3808
(70,700)	1.4221	1.5976	1.2947	1.5903	1.4873	1.6286	1.3685	1.6113	1.5059	1.6312	1.4039	1.6568	1.3922	1.6736	1.4106
(80,800)	1.4508	1.6474	1.3361	1.6364	1.5224	1.6704	1.4229	1.6706	1.5411	1.6771	1.4277	1.6827	1.4319	1.7288	1.4475
(90,900)	1.4269	1.6225	1.3231	1.6266	1.5027	1.6472	1.3988	1.6500	1.5283	1.6488	1.4353	1.6707	1.4387	1.7187	1.4362
(100,1000)	1.4038	1.5833	1.3284	1.6102	1.4578	1.5944	1.4200	1.6498	1.4652	1.592	1.3913	1.6132	1.3953	1.6699	1.4088

TABLE V

THRESHOLD BASED HEURISTIC EFFICIENCY.

and α_k parameters achieving in the worst case values equal to 1.59 and 1.73, respectively. These results show how performing the resource provisioning through our generalized Nash game allows obtaining better and more robust results with respect to heuristics based on utilization thresholds.

B. Algorithms Scalability

The scalability of our approach has been evaluated performing tests 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. KNITRO 7.0 has been use as non linear optimization solver. We have considered a very large set of randomly generated instances obtained as in the previous Section, varying the number of WS applications between 1,000 and 10,000, while guaranteeing that constraint (7) is active. We compare our algorithms with the solution we proposed in [10], where the social welfare is obtained by solving the generalized Nash game through its corresponding Variational Inequality (VI) relying on a projection method (see [10] for further details). Results are reported in Table VI, where also in this case, the figures are the means computed on ten different runs.

Overall Algorithm 1 performs better than Algorithm 2 solving on average almost $|\mathcal{P}|$ instead of $2\,|\mathcal{P}|$ optimization problems. However, Algorithm 2 is suitable of a fully distributed solution while Algorithm 1 has to be executed at the IaaS site. With respect to the VI solution, the speed up achieved shows that Algorithm 1 and 2 allows improving the overall execution time by almost one order of magnitude. Since the computation time in the worst case is less than one minute, our solutions are suitable to determine the resource provisioning of very large Cloud infrastructures on a hourly basis without introducing any system overhead.

$(\lvert\mathcal{P}\rvert, \lvert\mathcal{A}\rvert)$	Algorithm 1 Exe. time (sec.)	Algorithm 2 Exe. time (sec.)	Algorithm 1 speedup	Algorithm 2 speedup
(10,1000)	1.87	4.67	10.13	4.05
(20,2000)	4.03	8.70	33.17	15.37
(30,3000)	6.03	12.66	49.62	23.65
(40,4000)	8.30	17.21	13.92	6.71
(50,5000)	10.67	22.27	10.91	5.23
(60,6000)	13.30	27.55	9.45	4.56
(70,7000)	16.03	32.14	16.17	8.07
(80,8000)	18.63	37.86	16.02	7.88
(90,9000)	21.53	43.76	8.66	4.26
(100,10000)	24.63	49.37	10.47	5.23

TABLE VI

ALGORITHM 1 AND 2 AVERAGE EXECUTION TIME AND SPEEDUP WITH RESPECT TO THE SOLUTION PROPOSED IN [10].

C. Equilibria Sharing Analysis

The aim of this Section is to analyse how the on spot VMs are shared among competing SaaS providers changing the game parameters. The results have been obtained by Algorithm 2 only since the two solutions proposed do not differ significantly in terms of equilibria efficiency and execution time. Furthermore, Algorithm 2 is suitable for a fully distributed implementation.

In particular we considered three SaaSs offering five heterogeneous applications each. If not differently stated we set $s^U = 30$, $\varphi_k = 0.1\$$, $\delta_k := 0.11\$$, $h_p^U = 20$, $f_p^U = 10$, $\Lambda_k = 1$ req/sec, $\mu_k = k$ req/sec, $m_k = -1$, $\sigma_k^L = 0.03\$$, and $\sigma_k^U = 0.09\$$. In the following we will vary one parameter at the time for the first application $k = 1$, while the parameters of the remaining ones will be held fixed. Figures 5-8 show how the number of resources devoted to the first application (in terms of in house, flat, on demand, and on spot instances) and the overall capacity allocated to the remaining classes change as a function of the varying parameter. In particular, in Figure 5 the incoming workload Λ_1 varies between 1 and 16 req/sec. As the Figure shows, all of the in house instances are always used, and, as the workload increases, they are migrated from the other applications to application 1. In order to profitably sustain the workload, the number of on spot, flat, and on demand instances is also increased, when Λ_1 is around 3.7 req/sec, 11 req/sec, and 13.9 req/sec, respectively. As Λ_1 increases the resource allocation policy saturates the cheapest resource first. In general the resource allocation trends are linear with Λ_1, but are non-smooth. This is due to the fact that the equilibrium is not unique and hence the same performance and revenues can be obtained with multiple values of (h_k, f_k, d_k, s_k) (recall also that the social welfare function is concave but not strictly concave).

Figure 6 shows the resource sharing at the equilibrium changing the slope of application 1 utility function (which has been varied in the range $[-50, -1]$, we also set $\mu_1 = 0.25$ req/sec). As in the previous analysis, the in house capacity is migrated to application 1 which becomes more sensitive to response time variations and hence requires additional capacity. Again, cheaper resources are used first until their saturation (which is obtained for $m_1 = -10$ for on spot VMs, and $m_1 = -16$ for flat VMs). Also in this case the resource allocation trends are linear with $|m_1|$.

Figure 7 analyses how the GNE changes by varying application 1 maximum service rate (the range $[0.02, 1]$ req/sec has been considered). If the maximum service rate increases the service time required to process each application 1 request decreases and the overall capacity required to process application 1 decreases accordingly. Hence, in this case on spot instances are migrated

from application 1 to the other classes and on demand instances are used only when application 1 requests are very CPU intensive ($\mu_1 < 0.09$ req/sec).

Finally, Figure 8 shows how the equilibrium changes by varying the maximum time unit cost for application 1 (σ_1^U has been varied in the range $[0.02, 0.19]$\$, while for the remaining classes we set $\sigma_k^L = 0.01$\$ and $\sigma_k^U = 0.02$\$). We considered $\mu_1 = 0.05$. As σ_1^U increases the number of on spot VMs allocated to application 1 decreases since the IaaS set $\sigma_1 = \sigma_1^U$ and the SaaS provider can use in a more cost efficient way the on spot VMs to serve his remaining applications, while application 1 is supported by in house, flat and on demand instances. The number of on demand resources varies abruptly when $\sigma_1^U = 0.11$\$, i.e. when $\sigma_1^U = \delta_1$. This is very unintuitive, since increasing the maximum time unit cost one is willing to pay for a given application implies that the number of on spot instances devoted to the same application is reduced.

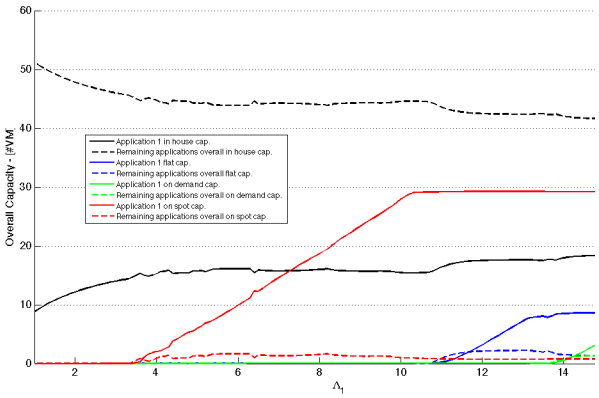

Fig. 5. Resource allocation with varying applicati... incoming workload.

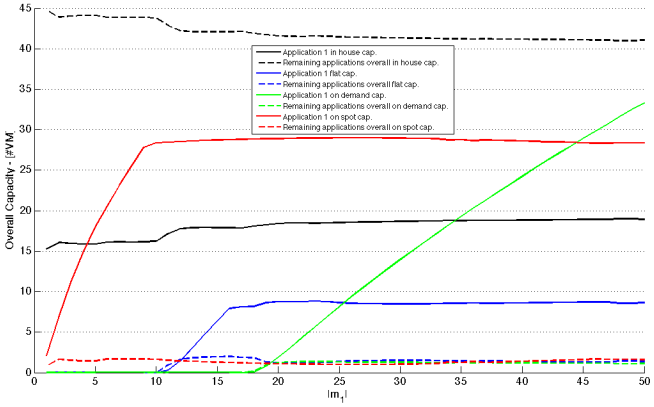

Fig. 6. Resource allocation with varying application 1 utility function slope.

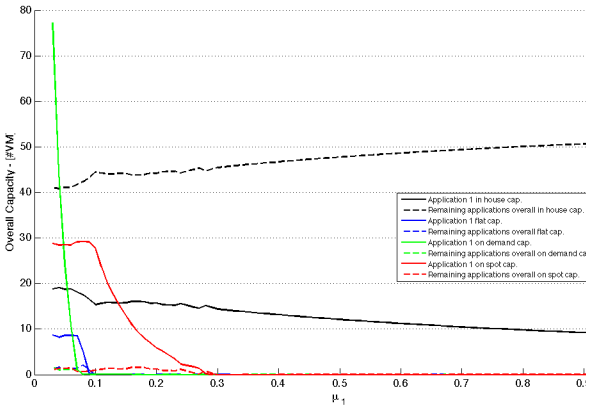

Fig. 7. Resource allocation with varying application 1 maximum service rate.

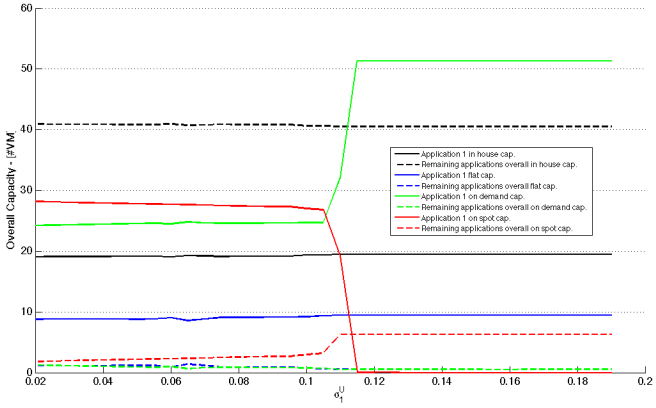

Fig. 8. Resource allocation with varying application 1 on spot maximum time unit cost.

D. 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 [40] benchmark. SPECweb2005 is the industry standard benchmark for the performance assessment of Web servers. We have considered the *e-commerce* and *banking* workloads, which simulate the access to an on-line trading and to an on-line banking Web site implementing an HTTPS/HTTP mix and HTTPS only load, respectively.

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[3]. 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 deploying VM server instances in Virginia, while the client components have been deployed in the North California Amazon region. We have obtained an estimate of the maximum service rate parameters and the network delay by performing an extensive off-line profiling along the lines of [37], [29] collecting a set of statistics and minimizing the mean square error for the response time. In particular we got $D_1 = 2.00$ sec and $\mu_1 = 222.26$ req/sec for the e-commerce workload, while we obtained $D_2 = 1.04$ sec and $\mu_2 = 281.67$ req/sec for banking. The maximum percentage error on the average response time estimation is less than 18%. We set $\alpha_1 = \alpha_2 = 0.4$, $m_1 = -1$, $m_2 = -10$, $\overline{R}_1 = 2.1$ sec, $\overline{R}_2 = 1.1$ sec, $\sigma_1^U = 0.25$\$, $\sigma_2^U = 0.30$\$, and considered the two workloads as two different applications offered by two independent SaaS providers.

Four our validation we considered realistic incoming workloads, created from a trace of *real* requests to a large Web system in Italy. The real system includes almost 100 servers and the

[3]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, $N_k = (R_k + Z_k) \cdot \Lambda_k$).

trace contains the number of sessions, on a per-hour basis, over a one year period. Incoming workloads are built as in [2], [11], assuming that the request arrivals follow non-homogeneous Poisson processes with rates changing every hour according to the trace. Data collected from the log on a hourly basis have been interpolated linearly and oversampled adding also some noise to obtain workload traces varying every 10 minutes as in [32]. The plot in Figure 9 reports the number of users during the experiment for the two workloads. We considered the peak hours (10.00-19.00) gathered from the log trace during the days which lead to the highest workload over the year. During the experiment only on demand and on spot instances have been considered. The numbers of VMs are determined every hour by Algorithm 2 and are plotted in Figure 10. For each application, since multiple VMs are used, the load is evenly shared among multiple instances by registering the VMs with an Amazon Elastic Load Balancer [7].

Figures 11 and 12 show the overall traffic served during the experiment, while Figures 13 and 14 report the end users average response time sampled every 10 sec. Results show that our resource allocation policies are effective, since the average response times are almost always below the thresholds.

VI. Related work

The recent development of Cloud systems and the rapid growth of the Internet have led to a remarkable development in the use of the Game Theory tools. Problems arising in the ICT industry, such as resource or quality of service allocation problem, pricing, and load shedding, can not be handled with classical optimization approaches. Indeed, in a pure optimization approach the goal of, in general, *a complex system* does not depend on the interrelationships among different users, or players [6]. However, interaction across different players is non-negligible: Each player can be affected by the actions of all players, not only by his own actions. In this setting, a natural modelling framework involves seeking an equilibrium, or stable operating point for the system. More precisely, each player seeks to optimize his own goal, which depends on the strategies of the other players upon his own, and this optimization is performed simultaneously by different players. An equilibrium (in the sense of Nash) is reached when no player can decrease his objective function by changing unilaterally its strategy.

A survey of different modelling and solution concepts of networking games, as well as a number of different applications in telecommunications and wireless networks, based on Game Theory, can be found in [44], [6].

With respect to telecommunication applications, a rich literature exists which includes solutions for flow and congestion control [33], [19], network routing [5], [15], file allocation [31], load

balancing [28], [27], multi-commodity flow [45], resource allocation [47], [24] and quality of service provisioning [21]. In the area of distributed computing systems, [23] the static load balancing problem in heterogeneous distributed systems is formulated as a non-cooperative game among users. Based on the Nash equilibrium concept, the authors derive a distributed load balancing algorithm, whose performance are compared with other existing schemes. The main advantages of the proposed solution are the distributed structure, low complexity and optimality of allocation for each user. Authors in [9] analysed the impact of non-cooperative users in a system of multiple parallel non-observable queues by studying the PoA under different workload and system configurations. The PoA has been evaluated recently for a wide spectrum of applications ranging from network routing [38] to cluster management [25], load balancing and scheduling [4], [17].

Regarding Cloud computing, the use of Game Theory for the resource allocation problem in Cloud environment is investigated in [42]. Here, the authors start from the bid proportional auction resource allocation model proposed in [16], and evolve the model from perfect information to an incomplete common information where one bidder does not know how much the others would like to pay for the computing resource. To this end a Bayesian learning mechanism is introduced, and it is shown that a Nash equilibrium solution exists among all the possibilities prices, which means that no one can get a better benefit without damaging others.

In [3], the authors consider centralized and decentralized load balancing strategies in a system with multiple and heterogeneous processor sharing servers. Each server has an associated service capacity and a holding cost per unit time. The requests arrive as a Poisson process, and the service time of incoming jobs is assumed to be known. For such system, the load balancing problem is investigated in two different scenarios: (i) a centralized setting leading to a global optimization problem, in which a dispatcher decides where each job will get service so as to minimize the weighted mean number of jobs in the system, and (ii) a distributed non-cooperative setting leading to a non-cooperative game transformed into a standard convex optimization problem. The paper studies structural properties of both strategies, and the efficiency loss in terms of PoA of the decentralized scheme relative to the global optimal (centralized) one.

In [48] the authors propose a pricing mechanism for allocation capacity in a utility computing system among competing end-users requests. The fixed available service capacity is allocated among the different flows proportionally to their monetary bids. The paper studies the resulting equilibrium point, establishes convergence of a best-response algorithm, and bounds the efficiency

loss (PoA) of this distributed mechanism. More precisely: End-users requests are represented as job flows in a controlled queueing system. These jobs arrive to the system through a fixed, random process, are stored in a buffer, and then are serviced by the resource in a first come, first served manner. The service rate is set through a proportional share mechanism. Within this framework, the interactions between end-users are modelled as a game. Then, authors show that the equilibrium can be reached in a distributed, asynchronous manner. The paper also reports the sensitivity analysis with respect to the variation of problem's parameters (e.g., load intensity and relative importance of the competing user requests). Differently from our point of view, in [48] the problem of the capacity allocation is considered for a single virtualized server among competing user requests, while in this paper we consider the infrastructure data center at a higher granularity (i.e., VMs).

In this paper we extend our work we presented in [10], by extending the Game model including also hybrid Cloud infrastructures. Furthermore, we proposed new algorithms able to determine the GNE up to one order of magnitude faster than our previous solution, without introducing significant efficiency loss in terms of PoA and IWC and suitable for a fully distributed implementation. A more in depth analysis of the proposed solution has also been performed, validating our resource allocation policies in a real prototype environment.

VII. Conclusions

We proposed a game theory based approach for the run-time management of a IaaS provider capacity among multiple competing SaaSs. The cost model consists of a class of utility functions which include revenues and penalties incurred depending on the achieved performance level and the infrastructural costs associated with IaaS resources. The solution is effective even for very large size problem instances. Systems up to thousands of applications can be managed very efficiently. The effectiveness of our approach has been assessed by performing simulation and experiments in a real prototype environment. Synthetic as well as realistic workloads and a number of different scenarios of interest have been considered. A comparison with utilization based state-of-the-art techniques shows that our solutions outperform alternative methods providing results up to 50-60% better in terms of equilibrium efficiency. Moreover, solutions are more robust to performance parameters settings and system configurations.

Future work will extend the proposed solution to consider multiple-time scales and including also a request redirect mechanism to share the workload among multiple Cloud sites.

ACKNOWLEDGEMENT

The experimentation on Amazon EC2 has been supported by the 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.

REFERENCES

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

[2] J. M. Almeida, V. A. F. Almeida, D. Ardagna, I. S. Cunha, C. Francalanci, and M. Trubian. Joint admission control and resource allocation in virtualized servers. *J. Parallel Distrib. Comput.*, 70(4):344–362, 2010.

[3] E. Altman, U. Ayesta, and B. Prabhu. Load balancing in processor sharing systems. In *ValueTools '08 Proc.*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008.

[4] E. Altman, U. Ayesta, and B. J. Prabhu. Optimal load balancing in processor sharing systems. In *In Proc. of GameComm*, 2008.

[5] E. Altman, T. Basar, T. Jimenez, and N. Shimkin. Competitive routing in networks with polynomial cost. *IEEE Trans. on Automatic Control*, 47(1):92–96, 2002.

[6] E. Altman, T. Boulogne, R. El-Azouzi, T. Jiménez, and L. Wynter. A survey on networking games in telecommunications. *Comput. Oper. Res.*, 33(2):286–311, 2006.

[7] Amazon Inc. Amazon Elastic Cloud. http://aws.amazon.com/ec2/.

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

[9] J. Anselmi and B. Gaujal. Optimal routing in parallel, non-observable queues and the price of anarchy revisited. In *Teletraffic Congress (ITC), 2010 22nd International*, 2010.

[10] D. Ardagna, B. Panicucci, and M. Passacantando. A Game Theoretic Formulation of the Service Provisioning Problem in Cloud Systems. In *WWW 2011 Proc. To Appear*, 2011.

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

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, I. S. A. Rabkin, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.

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

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

[15] T. Boulogne, E. Altman, H. Kameda, and O. Pourtallier. Mixed equilibrium in multiclass routing games. *IEEE Trans. on Automatic Control*, 47(6):903–916, 2002.

[16] J. Bredin, D. Kotz, D. Rus, R. Maheswaran, C. .Imer, and T. Basar:. Computational markets to regulate mobile-agent systems. *Autonomous Agents and Multi-Agent Systems*.

[17] H.-L. Chen, J. R. Marden, and A. Wierman. The effect of local scheduling in load balancing designs. *SIGMETRICS Perform. Eval. Rev.*, 36:110–112, August 2008.

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

[19] W. Chink. On convergence of asynchronous greedy algorithm with relaxation in multiclass queuing environment. *IEEE Communication Letters*, 3:34–36, 1999.

[20] G. Debreu. A social equilibrium existence theorem. *Proc. of the Nat. Academy of Sciences of the USA*, 38:886–893, 1952.

[21] R. El-Azouzi and E. Altman. Constrained traffic equilibrium in routing. *IEEE/ACM Trans. on Automatic Control*, 48(9):1656–1660, 2003.

[22] F. Facchinei and C. Kanzow. Generalized Nash equilibrium problems. *Ann. Oper. Res.*, 175:177–211, 2010.

[23] D. Grosu and A. Chronopoulos. Noncooperative load balancing in distributed systems. *J. Parallel Distrib. Comput.*, 65(9):1022–1034, 2005.

[24] M. Haviv. The aumann-shapely pricing mechanism for allocating congestion costs. *Operations Research Letters*, 29(5):211–215, 2001.

[25] M. Haviv and T. Roughgarden. The price of anarchy in an exponential multi-server. *Oper. Res. Lett.*, 35(4):421–426, 2007.

[26] J. Hamilton. Using a Market Economy. http://perspectives.mvdirona.com/2010/03/23/UsingAMarketEconomy.aspx.

[27] H. Kameda, E. Altman, T. Kozawa, and Y. Hosokawa. Braess-like paradoxes in distributed computer systems. *IEEE Trans. on Automatic Control*, 45(9):1687–1691, 2000.

[28] H. Kameda, J. Li, C. Kim, and Y. Zhang. *Optimal load balancing in distributed computer systems*. Berlin: Springer, 1997.

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

[30] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan. vManage: loosely coupled platform and virtualization management in data centers. In *ICAC2009 Proc.*, 2009.

[31] J. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Trans. on Computers*, 38(5):705–717, 1989.

[32] D. Kusic, J. O. Kephart, N. Kandasamy, and G. Jiang. Power and Performance Management of Virtualized Computing Environments Via Lookahead Control. In *ICAC 2008 Proc.*, 2008.

[33] R. Mazumdar, L. Mason, and C. Douligeris. Fairness in network optimal flow control: optimality of product forms. *IEEE Trans. on Communications*, 39(5):775–782, 1991.

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

[35] D. Monderer and L. Shapley. Potential games. *Games and Economic Behaviour, Elsevier*, 14:124–143, 1996.

[36] J. Nash. Non-cooperative games. *Annals of Mathematics. Second Series*, 54:286–295, 1951.

[37] 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.

[38] T. Roughgarden. The price of anarchy is independent of the network topology. In *STOC*, pages 428–437, 2002.

[39] T. Roughgarden. Algorithmic game theory. *Commun. ACM*, 53(7):78–86, 2010.

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

[41] SpotHistory.com. Spot Instance Price History Graphs. http://www.spothistory.com/.

[42] F. Teng and F. Magoules. A new game theoretical resource allocation algorithm for cloud computing. In *Advances in Grid and Pervasive Computing*, pages 321–330, 2010.

[43] B. Urgaonkar and P. Shenoy. Sharc: Managing CPU and Network Bandwidth in Shared Clusters. *IEEE Trans. on Parallel and Distr. Systems*, 15(1):2–17, 2004.

[44] A. van den Nouweland, P. Borm, W. van Golstein Brouwers, R. Bruinderink, and S. Tijs. A game theoretic approach to problems in telecommunication. *Manage. Sci.*, 42(2):294–303, 1996.

[45] M. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

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

[47] H. Yaiche, R. Mazumdar, and C. Rosenberg. A game theoretic framework for bandwidth allocation and pricing of elastic connections in broadband networks. *IEEE/ACM Trans. on Networking*, 8(5):667–678, 2000.

[48] B. Yolken and N. Bambos. Game based capacity allocation for utility computing environments. In *ValueTools '08 Proc.*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2008.

[49] 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.

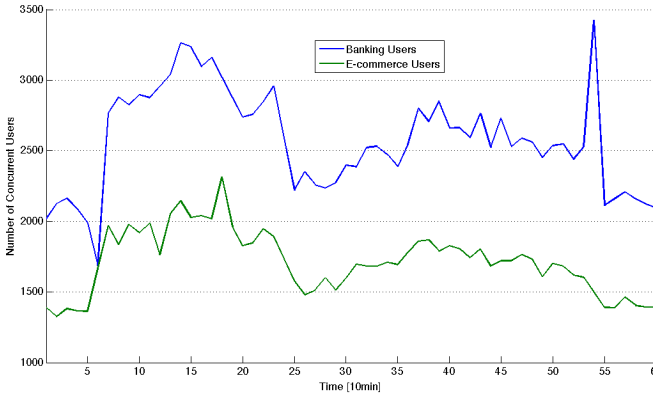

Fig. 9. Number of users used in the EC2 experiment.

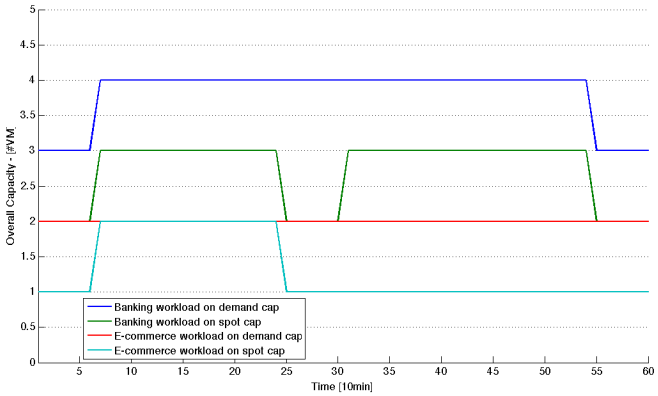

Fig. 10. VM instances used in the EC2 experiment.

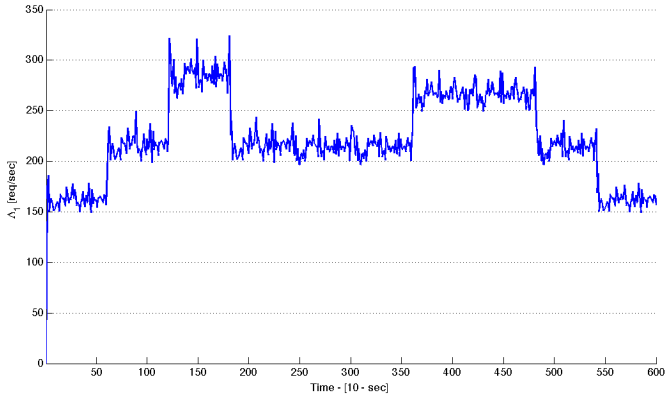

Fig. 11. Overall workload served for the e-commerce workload.

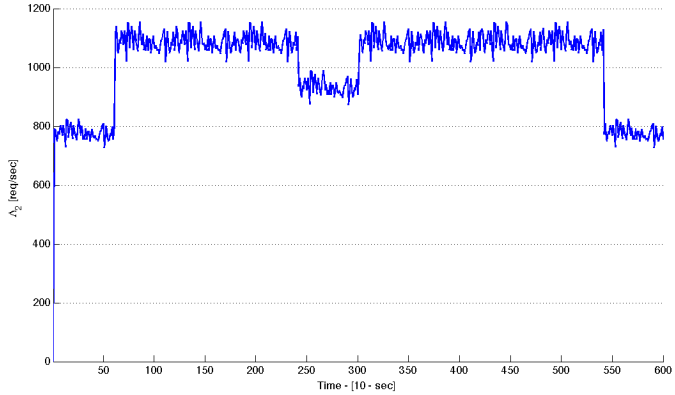

Fig. 12. Overall workload served for the banking workload.

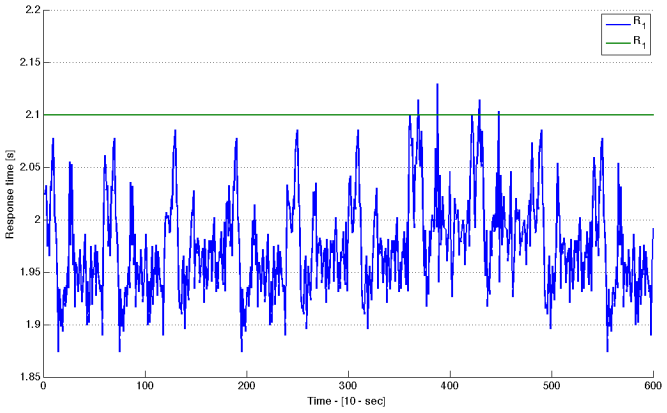

Fig. 13. Average response time measured for the e-commerce workload.

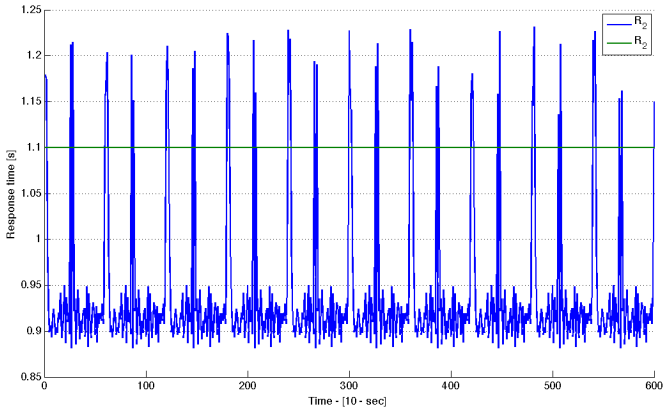

Fig. 14. Average response time measured for the banking workload.