Generalized Nash Equilibria for the Service Provisioning Problem in Multi-Cloud Systems

D. Ardagna[1], M. Ciavotta[1], and M. Passacantando[2]

[1]Dipartimento di Elettronica, Informazione e Bioingegneria Politecnico di Milano, Milan, Italy
[2]Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Abstract—The adoption of Cloud technologies is steadily increasing. In such systems, applications can benefit from nearly infinite virtual resources on a pay-per-use basis. However, being the Cloud massively multi-tenant and characterized by highly variable workloads the development of more and more effective provisioning policies assumes paramount importance. Boosted by the success of the Cloud, the application of Game Theory models and methodologies has also become popular, since they have been demonstrated to suit perfectly to Cloud social, economic, and strategic structures. This paper aims to study, model and efficiently solve the cost minimization problem associated with the service provisioning of SaaS virtual machines in multiple IaaSs. We propose a game-theoretic approach for the runtime management of resources from multiple IaaS providers to be allocated to multiple competing SaaSs, along with a cost model including revenues and penalties for requests execution failures. A distributed algorithm for identifying Generalized Nash Equilibria has been developed and analysed in detail. The effectiveness of our approach has been assessed by performing a wide set of analyses under multiple workload conditions. Results show that our algorithm is scalable and provides significant cost savings with respect to alternative methods (80% on average). Furthermore, increasing the number of IaaS providers SaaSs can achieve 9–15% cost savings from the workload distribution on multiple IaaSs.

Index Terms—Cloud Computing; Game Theory; Generalized Nash Equilibrium.

1 Introduction

In recent years we are witnessing the gradual virtualization of software systems. We come from a world where applications were entirely developed in-house, possibly exploiting third-party components and/or frameworks, but mainly deployed and executed within each organization private IT facilities. With the advent of Service Oriented systems, applications were rethought as a collection of interacting services, in some cases delegating part of their functionalities to existing external software services provided by other organizations. The recent advent of Cloud computing rewrote the rules of software development by pushing for a greater use of virtualization. Cloud users can now access not only to third-party software components (Software as a Service - SaaS), but also to fully virtualized hardware resources (Infrastructure as a Service - IaaS) or even to complete development and execution environments (Platform as a Service - PaaS), paying only for the resources actually used. The adoption of Cloud computing is attractive: users obtain Cloud resources, whose management is partly automated and can be scaled almost instantaneously. However, Cloud technologies are still characterized by open critical issues. Specifically, Cloud performance shows a high variability in time, elasticity might not ramp at the desired speed and unavailability problems do exist even when 99.9% up-time is advertised (see, e.g., the Microsoft Azure outage in 8th August 2014 [32]). Moreover, modern Cloud applications operate in an open and dynamic world characterized by constant, often unpredictable, changes. In such a context, the development of efficient *service provisioning* policies, possibly spread among multiple IaaS providers [14], [49], is very challenging.

If we consider Internet systems, *non-cooperative Game Theory models* have been successfully applied to diverse problems such as flow and congestion control, routing, and networking, but also resource allocation and pricing. Game theory approaches allow to gain an in-depth analytical understanding of problems that can not be handled with classical optimization approaches because each player can be affected by the actions taken by all the other players involved, not only by his/her own decisions [6]. One of the most widely used solution concept in Game Theory is the Nash Equilibrium [50]: a set of strategies constitutes a Nash Equilibrium if no player can benefit by changing his/her strategy unilaterally or, in other words, every player is playing a *best response* to his/her opponents' strategy choices.

In this work, we develop novel resource allocation policies for Cloud systems based on the formulation and a sound solution of a game theoretical model. We take the perspective of SaaS providers that host their applications at multiple IaaS providers, thanks to a software middleware developed within the framework of the MODAClouds project [14], [49]. Each SaaS provider aims at minimizing the usage cost of Cloud resources and incurs penalties in case of request execution failures. The cost minimization is challenging since online services receive variable workloads during the day. Furthermore, each SaaS behaves selfishly and competes with others SaaSs for the use of the infrastructural resources supplied by the IaaSs. Each IaaS, in turn, wants to maximize the revenues obtained providing the resources. To capture the behavior of SaaSs and IaaSs

danilo.ardagna@polimi.it
michele.ciavotta@polimi.it
mauro.passacantando@unipi.it

in this conflicting situation, in which the best choice for one depends on the choices of the others, we model such problem as a *Generalized Nash Equilibrium Problem (GNEP)*, in which both the objective function and the feasible region of each player depend on the strategies chosen by the other players (see, e.g., [23], [27], [29], [56]). Then, we propose a distributed algorithm which converges to a Generalized Nash Equilibrium after a finite number of iterations. Experimental campaigns show that our algorithm is scalable and provides significant cost savings with respect to alternative methods currently adopted in real systems (80% on average). Furthermore, by increasing the number of IaaS providers we show that SaaSs can achieve 9-15% cost savings from the workload distribution on multiple IaaSs. A viability analysis of the data synchronization in terms of cost associated with the egress traffic in multi-IaaS deployments is also provided, along with a study of the bandwidth requirements of the proposed distributed algorithm.

The remainder of the paper is organized as follows. A review of other approaches presented in literature is provided in Section 2. The runtime service provisioning problem and some related design assumptions are introduced in Section 3. The problem under study is modeled as a GNEP in Section 4 and an analysis of its properties is presented in Section 5. In Section 6, we develop an efficient algorithm for the runtime management and allocation of IaaSs resources to competing SaaSs suitable also for a *fully distributed* implementation. Section 7 is devoted to assess the quality of our solution through analyses and experiments. Conclusions are finally drawn in Section 8.

2 Related Work

In recent years we witnessed the rise and consolidation of Cloud systems. Consequently, the application of Game Theory models and methodologies has also increased, since they have been demonstrated to suit perfectly to the social, economic and strategic structure of the Cloud [6]. In Clouds, interaction across different players is non-negligible: each player (e.g., IaaS or SaaS provider or individual end-users) can be affected by the actions of all players, not only by its own actions. Game Theory can reproduce perfectly this aspect. In this setting, a natural modeling framework involves seeking an equilibrium, or stable operating point for the system.

Game Theory solutions for resource management and load balancing of distributed computing systems are presented in [5], [10], [24], [39], [59], while a survey on networking games, as well as a number of different applications in telecommunications and wireless networks, can be found in [6]. As for the specific issues of Cloud Computing, various theories and methods have been used to represent, model and manage Cloud services. Different types of equilibria and games have been considered, according to the problem addressed. A detailed survey of several resource allocation strategies in Clouds is given in [64], whereas a recent comparative study can be found in [40].

In [62] the authors investigate the use of Game Theory for the resource allocation problem in Cloud environments starting from the bid-proportional auction model for resource allocation proposed in [20]. An incomplete common information model is presented, in which one bidder does not know how much the others would like to pay for the computing resource. A bid-proportional auction model is also presented in [63] to adaptively tune the price of Cloud resources. The proposed model is validated through simulation by considering dynamic and stochastic demands. A pricing mechanism for allocation capacity in a utility computing system among competing end user requests is proposed in [68], where the fixed available service capacity is allocated on the different flows proportionally to their monetary bids. However, the capacity allocation problem is considered for a single virtualized server, while in this work we consider the whole data center infrastructure.

In [31] the authors present a methodical in-depth game theory study on price competition, moving progressively from a monopoly market to a duopoly market, and finally to an oligopoly Cloud market. They characterize the nature of non-cooperative competition with multiple competing Cloud service providers, derive algorithms representing the influence of resource capacity and operating costs on the solution and prove the existence of a Nash equilibrium. Studies on the maximization of the social welfare as a long-term social utility are discussed in [46]. Considering relevant queuing aspects in a centralized setting, the work establishes existence and uniqueness of the social optimum.

Two simple pricing schemes for selling Virtual Machine (VM) instances and the trade-off between them are studied in [1]. Exploiting Bayes Nash equilibrium, the authors provide theoretical and simulation based evidences suggesting that fixed prices generate a higher expected revenue than hybrid systems. Another work regarding spot bidding is presented in [60]: the authors propose a profit aware dynamic bidding algorithm, which observes the current spot price and selects bids adaptively to maximize the average profit of a Cloud service broker, while minimizing costs in a spot instance market. Similarly, a bid selection algorithm for the optimal capacity segmentation problem with hybrid pricing aimed at maximizing the revenue of a Amazon EC2-like provider can be found in [65].

A game theoretic model of IaaS Cloud market that includes price dynamics, usage, load profiles and economy of scale in hybrid Cloud scenario is proposed in [43]. Dynamic resource pricing of multiple geo-distributed Cloud providers interacting with multiple competing application providers is formulated in [55] as a Stackelberg game. The cost minimization of dynamic service placement in a geographically distributed Cloud, while fulfilling at the same time certain Quality of Service (QoS) requirements, is faced in [70]. Authors present a dynamic solution based on control models and study how the price fluctuate in the presence of several customers competing for Cloud resources. A non-cooperative game is presented for which a Nash equilibrium exists that is also socially optimum.

Three Cloud resource procurement approaches based on economic and game-theoretic models are presented in [53]. The authors demonstrate that the larger the number of vendors, the lower procurement costs tend to be, irrespective to the considered approach, and also provide a mechanism to select a suitable Cloud vendor in a multi-IaaS market.

Similarly, three Cloud network economic problems are analyzed in [51] and modeled as non-cooperative price and QoS games among multiple Cloud providers.

A multi-SaaS/single-IaaS problem with flat, on demand and on spot VM instances is faced in [28], where a two stage provisioning mechanism is proposed. In the first stage, the VM requirements are calculated for each SaaS provider by means of standard optimization techniques, whereas the second stage, modeled as a Stackelberg game, determines the spot price that maximizes the IaaS revenue. The same problem is considered in [2], but the second stage is modeled as an N-armed bandit problem in which every player has to choose among N options taking into account past feedback.

The resource allocation problem among different competitive Cloud providers is considered in [66]: it is modeled as a competitive normal-form game for which the existence of a unique Nash equilibrium is proved.

Differently to previous literature proposals, we consider a more complex setting where the choice of each player influences the feasible strategy set of the others. This can be naturally modeled as a Generalized Nash Equilibrium Problem (GNEP), which is usually more difficult to solve than a standard Nash Equilibrium Problem [29].

We considered GNEPs for service provisioning problem in [12], [15], [16], where the perspective of SaaS providers hosting their applications at an IaaS provider is taken. Each SaaS needs to comply with end user Service Level Agreements (SLAs) and at the same time minimize the cost of use of resource supplied by the IaaS, whereas the IaaS wants to maximize the revenues obtained providing on spot resources. It is worth to be noticed that in this paper we extend our previous work modeling the multi-Saas/multi-IaaS situation and considering realistic follow-the-sun workloads that span over a 24-hour time horizon. In this context, the considered problem turns out to be remarkably more complex to analyze and to solve. Nonetheless, it is certainly an important problem that deserves to be considered as proven by the significant savings showed with respect to the multi-Saas/single-IaaS case, even in presence of data replication and synchronization costs.

3 Problem Statement and Assumptions

In this work, 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 and can be deployed at multiple IaaS providers.

We denote by $\mathcal{I}$ and $\mathcal{S}$ the sets of IaaSs and SaaSs, respectively, by $\mathcal{S}_i \subseteq \mathcal{S}$ the set of SaaS providers running at IaaS $i \in \mathcal{I}$ and by $\mathcal{I}_j \subseteq \mathcal{I}$ the set of IaaS providers supporting SaaS $j \in \mathcal{S}$. Moreover, let $\mathcal{A}$ be the overall set of WS applications, we denote by $\mathcal{A}_j \subseteq \mathcal{A}$ the set of WS applications offered by SaaS $j \in \mathcal{S}$, and by $\mathcal{A}_i \subseteq \mathcal{A}$ the set of WS applications running at IaaS $i \in \mathcal{I}$.

A SLA contract, associated with each WS application, is established between the SaaS provider and its end users and predicates on the average service response time. In particular, the average response time $E[R_k]$ for executing WS application k has to be less or equal to a given threshold $\overline{R}_k$. Moreover, SaaS providers implement an admission control mechanism, i.e., incoming requests might be rejected. However, if a SaaS provider rejects a request it has to pay a penalty ν_k to its end user (specified again within the SLA) usually far higher than the related processing cost.

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 1), which is common for current Cloud solutions.

Applications are hosted in VMs that are dynamically instantiated by IaaS providers up to a maximum of number equal to N_i for any $i \in \mathcal{I}$. We assume that VMs are homogeneous within an IaaS provider, and heterogeneous across different providers. Moreover, we assume that SaaS providers can run their WS applications concurrently among multiple IaaSs that compete each other in the Cloud market, and we consider the runtime provisioning of resources at individual IaaS. This approach is possible thanks to a software layer developed by the MODAClouds project [14], [49], which allows concurrent execution, runtime migration and data synchronization of applications among multiple Cloud providers. MODAClouds provides overall solutions for the modeling, deployment, and runtime management of multi-Cloud applications. It aims at removing all limitations or technological lock-ins that currently prevent the execution of a distributed application on different Clouds. In our work, we will demonstrate that deploying applications on multiple Clouds increases applications availability and implies cost savings for SaaS providers which can benefit from IaaS competition and redistribute workload at runtime to the cheapest IaaSs. From a technological perspective, concurrent application execution on multiple clouds requires to synchronize data among database replica (either relational or NoSQL) hosted on different Clouds. For this purpose, MODAClouds runtime platform provides also a distributed middleware solution (see Figure 1) in charge of synchronizing data among (even technological) different databases [57]. We denote with $z_{k\,i_1\,i_2}$ the data synchronization rate from IaaS i_1 to IaaS i_2 for WS application k. In striving for simplicity, we decided not to introduce synchronization costs in the game formulation. However, a study on the impact of data replication cost on SaaS savings due to multi-IaaS deployment is presented in Section 7.5 along with a possible extension of the game.

IaaS providers usually charge the use of their resources on an hourly basis. Hence, the SaaS has to face the problem of determining every hour the optimal number of VMs for each WS application in order to minimize costs and penalties, performing resource allocation on the basis of a prediction of future WS workloads Λ_k. Concerning such predictions (which formally can be modelled as random variables), several methods have been adopted in many fields for decades [22] (e.g., ARMA models, exponential smoothing, and polynomial interpolation), making them suitable to forecast seasonal workloads, common on hourly basis, as well as non-stationary request arrivals characterizing more fine grained time scales [3]. In general, each prediction mechanism is characterized by several alternative

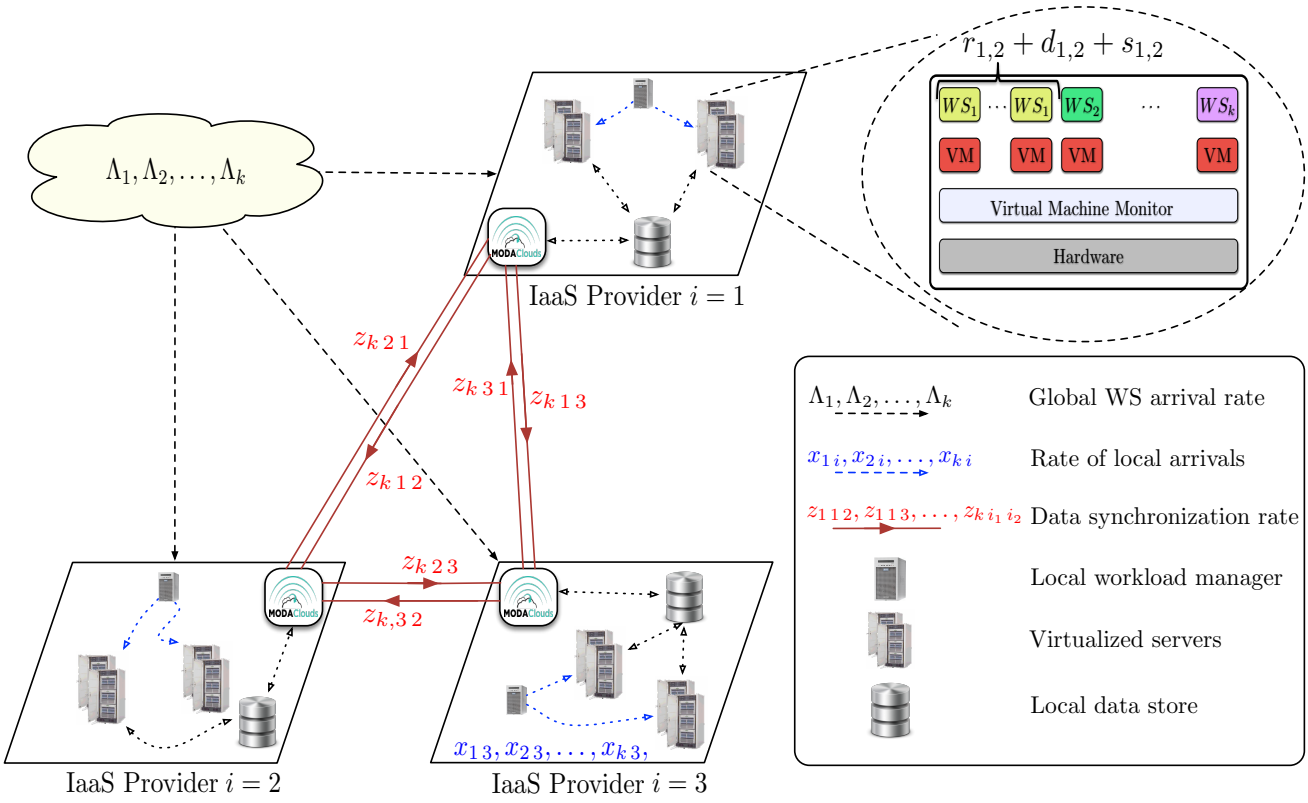

Figure 1. Cloud infrastructures and data migration and synchronization system

implementations, where the choice about filtering or not filtering input data (i.e., a run-time measure of the metric to be predicted) and choosing the best model parameters in a static or dynamic way are the most significant.

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 application hosted in a VM as an M/G/1 queue (see Figure 2) in tandem with a delay center, as in [41], [69]. We assume (as common among Web service containers) that requests are served according to the processor sharing scheduling discipline [3]. The delay center allows to model network delays and/or protocol delays introduced in establishing connections, etc. Performance parameters are also continuously updated at runtime in order to capture transient behavior, VMs network and I/O interference and performance time of the day variability of the Cloud provider (see [69] for further details). We denote with μ_{ki} and D_{ki} the maximum service rate and queueing delay for executing WS application k at IaaS i, respectively (see Figure 2). Although more accurate performance models exist in the literature for Web and Cloud systems (e.g., [21], [38], [54]), there is a fundamental trade-off between the accuracy of the models and their mathematical tractability, which has prevented us from exploiting such results here. It is important to note, however, that highly accurate approximations for general queueing networks [33], [34], having functional forms similar to our performance model, can be directly incorporated into our modeling and game theoretical framework.

For the IaaS provider we consider a pricing model similar to Amazon EC2 [8]: each IaaS provider offers *reserved*

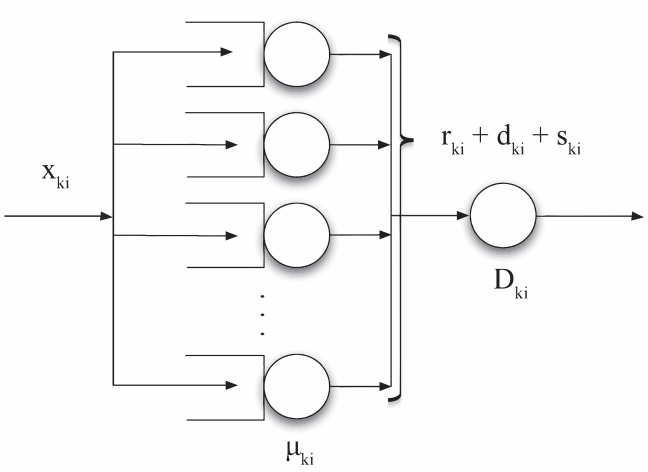

Figure 2. System performance model.

VMs, for which SaaS providers applies for a one-time payment (currently every one or three years) for each instance they want to reserve, *on demand* VMs, that let the SaaS pay for compute capacity by the hour with no long-term commitments, and *on spot* VMs, for which SaaS providers bid and compete for unused IaaS capacity.

In the following, we will denote with r_{ki}, d_{ki} and s_{ki} the number of reserved, on demand, and on spot instances for WS application k running at any time instant at IaaS i, respectively. VM instances are charged with the on spot cost σ_{ji} to the SaaS j by the IaaS i, σ_{ji} fluctuates periodically depending on the IaaS provider time of the day energy costs ω_i and on the supply of VMs and demand from SaaS for on spot VMs [8]. On spot costs vary also with the Cloud site

region, and we assume that SaaS providers set an upper bound σ_{ji}^{Max} strictly lower than the on demand VM time unit cost δ_i, because 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. On spot instances have been traditionally adopted to support batch computing intensive workload during peak periods, but we advocate their use also for traditional transactional services as in [15], [16]. The reserved instances time unit cost is equal to ρ_i, according to the SLA contract, and each SaaS j cannot have more than R_{ji} at IaaS i (for which SaaS applied for long-term contract). We denote with η_j the maximum fraction of resources allocated as on spot VMs for SaaS provider j to allow a reasonable reliability for every WS application. For the sake of clarity, the notation adopted here is summarized in Table 1.

4 Generalized Nash game model

The resource provisioning problem under study describes a conflicting situation, in which the optimal choices of SaaS and IaaS providers depend on the choices of the others. In this section, we formulate this problem as a Generalized Nash Equilibrium Problem. Section 4.1 is devoted to the formulation of the SaaS resource allocation problem, Section 4.2 presents the IaaS providers optimization problems, while the Generalized Nash equilibria of the game are defined in Section 4.3.

4.1 Game formulation from the SaaS side

The goal of SaaS provider j is to determine every hour the number of reserved r_{ki}, on demand d_{ki}, on spot VMs s_{ki} and the throughput x_{ki} in order to minimize its costs and, at the same time, to satisfy the prediction Λ_k for the arrival rate of the WS application k running at IaaS i to avoid the risk of paying penalties.

If the workload is evenly shared among the VMs, then the average response time for application k requests execution is given by (see Figure 2):

$$D_{ki} + \frac{1}{\mu_{ki} - \frac{x_{ki}}{r_{ki}+d_{ki}+s_{ki}}}.$$

We further assume that the VMs are not saturated, i.e., the equilibrium conditions for the M/G/1 queues hold:

$$\mu_{ki}\,(r_{ki} + d_{ki} + s_{ki}) - x_{ki} > 0.$$

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

$$\min_{r_{ki},d_{ki},s_{ki},x_{ki}} \Theta_j = \sum_{k\in\mathcal{A}_j}\sum_{i\in\mathcal{I}_j}(\rho_i\, r_{ki} + \delta_i\, d_{ki} + \sigma_{ji}\, s_{ki}) + \sum_{k\in\mathcal{A}_j} T\,\nu_k\,(\Lambda_k - X_k) \quad (1)$$

subject to the constraints:

$$D_{ki} + \frac{1}{\mu_{ki} - \frac{x_{ki}}{r_{ki}+d_{ki}+s_{ki}}} \le \overline{R}_k, \qquad \forall\, k \in \mathcal{A}_j, \forall\, i \in \mathcal{I}_j, \quad (2)$$

$$x_{ki} < \mu_{ki}\,(r_{ki} + d_{ki} + s_{ki}), \qquad \forall\, k \in \mathcal{A}_j, \forall\, i \in \mathcal{I}_j, \quad (3)$$

$$s_{ki} \le \frac{\eta_j}{1-\eta_j}\,(r_{ki} + d_{ki}), \qquad \forall\, k \in \mathcal{A}_j, \forall\, i \in \mathcal{I}_j, \quad (4)$$

$$\sum_{k\in\mathcal{A}_j} r_{ki} \le R_{ji}, \qquad \forall\, i \in \mathcal{I}_j, \quad (5)$$

$$\sum_{j\in\mathcal{S}_i}\sum_{k\in\mathcal{A}_j}(r_{ki} + d_{ki} + s_{ki}) \le N_i, \qquad \forall\, i \in \mathcal{I}_j, \quad (6)$$

$$\sum_{i\in\mathcal{I}_j} x_{ki} = X_k, \qquad \forall\, k \in \mathcal{A}_j, \quad (7)$$

$$\lambda_k \le X_k \le \Lambda_k, \qquad \forall\, k \in \mathcal{A}_j, \quad (8)$$

$$r_{ki}, d_{ki}, s_{ki}, x_{ki} \ge 0, \qquad \forall\, k \in \mathcal{A}_j, \forall\, i \in \mathcal{I}_j. \quad (9)$$

The SaaS goal is to minimize its cost function (1) which includes the fees requested by the IaaSs for instances used and the penalties incurred when requests are discarded (note that $\sum_{k\in\mathcal{A}_j} T\,(\Lambda_k - X_k)$ is the total number of requests rejected in time T, i.e., one hour).

Constraint (2) ensures that the response time is less or equal to the threshold $\overline{R}_k$ established in the SLA contract, while (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 $\eta_j < 1$ of the total capacity allocated for WS application k at IaaS i. Constraints (5) and (6) entail that allocated VMs are less or equal to the maximum number reserved or available at IaaS providers. Constraint (5) depends only on applications run by SaaS j, while constraint (6) refers to all applications of the set of SaaSs running at IaaS i. In addition to response time constraints and the number of VMs due to IaaSs limited capacity, we have throughput constraints: constraints (7) defines the total traffic served by the system as a fraction of the one served by individual IaaSs. Furthermore, constraint (8) establish a lower bound λ_k for the total throughput needed to satisfy SLA contracts and an upper bound Λ_k equal to the total incoming workload.

We remark that, in the formulation of the problem, we have imposed variables r_{ki}, d_{ki} and s_{ki} to be non-negative but not integer, as in reality they are. In fact, requiring variables to be integer makes the solution much more difficult (NP-hard). However, 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 (see also [16]). We therefore decide to deal with continuous variables, actually considering a relaxation of the problem.

4.2 Game formulation from the IaaS side

On the other side, the goal of each IaaS provider i is to determine the time unit cost σ_{ji} for on spot VM instances of each SaaS provider j in order to maximize its total revenue:

$$\max_{\sigma_{ji}} \Theta_i = \sum_{j\in\mathcal{S}_i}\sum_{k\in\mathcal{A}_j}[(\rho_i - \omega_i)\, r_{ki} + (\delta_i - \omega_i)\, d_{ki} + (\sigma_{ji} - \omega_i)\, s_{ki}] \quad (10)$$

Parameters	
$\mathcal{S}$	Set of SaaS providers
$\mathcal{I}$	Set of IaaS providers
$\mathcal{S}_i$	Subset of SaaS providers set $\mathcal{S}$ running applications at IaaS $i \in \mathcal{I}$
$\mathcal{I}_j$	Subset of IaaS providers set $\mathcal{I}$ supporting SaaS $j \in \mathcal{S}$
$\mathcal{A}$	Set of applications of all the SaaS providers
$\mathcal{A}_j$	Subset of applications set $\mathcal{A}$ of the SaaS provider $j \in \mathcal{S}$
$\mathcal{A}_i$	Subset of applications set $\mathcal{A}$ running at IaaS $i \in \mathcal{I}$
Λ_k	Prediction of the arrival rate for application k
λ_k	Minimum arrival rate to be guaranteed for application k
μ_{ki}	Maximum service rate for executing application k at IaaS i
D_{ki}	Queueing delay for executing application k at IaaS i
$\overline{R}_k$	application k average response time threshold
ν_k	Penalty for rejecting a single application k request
ρ_i	Time unit cost for reserved VMs for SaaS providers at IaaS i
δ_i	Time unit cost for on demand VMs for SaaS providers at IaaS i
ω_i	VM time unit energy cost for IaaS provider i
η_j	Maximum fraction of total resources allocated as on spot VMs for SaaS provider j
N_i	Maximum number of VMs that can be executed at the IaaS i
R_{ji}	Maximum number of reserved VMs that can be executed for the SaaS j at IaaS i
σ_{ji}^{Max}	Maximum time unit cost offered by SaaS j for on spot VMs instances to IaaS i
T	Control time horizon
SaaS Decision Variables	
r_{ki}	Number of reserved VMs used for application k at IaaS i
d_{ki}	Number of on demand VMs used for application k at IaaS i
s_{ki}	Number of on spot VMs used for application k at IaaS i
x_{ki}	Throughput for application k at IaaS i
X_k	Overall throughput for application k
IaaS Decision Variables	
σ_{ji}	Time unit cost set by IaaS i to SaaS j for on spot VMs instances

Table 1
Parameters and decision variables.

subject to the following constraints:

$$\sigma_{ji} \geq \omega_i, \qquad \forall\, j \in \mathcal{S}_i, \tag{11}$$

$$\sigma_{ji} \leq \sigma_{ji}^{Max}, \qquad \forall\, j \in \mathcal{S}_i. \tag{12}$$

The on spot instance cost lower bound (11) is the energy cost for running a single VM instance for one hour according to the time of the day, while the upper bound (12) is a fixed value σ_{ji}^{Max} established by the SaaS j provider.

4.3 Generalized Nash Equilibria

In this framework, SaaS and IaaS providers are making decisions at the same time. For each SaaS provider, the objective function depends on the strategies of the IaaS providers and the feasible region on the strategies of others SaaSs (see constraint (6)), while the objective function of each IaaS depends on SaaS decisions. To capture the behavior of SaaSs and IaaSs in this conflicting situation (game), we model the problem as a Generalized Nash Equilibrium Problem (GNEP), which is broadly used in Game Theory and other fields [29]. We recall that a GNEP differs from the classical Nash Equilibrium Problem since, not only the objective function of each player depends upon the strategies chosen by all the other players, but also each player's strategy set may depend on the rival players' strategies.

The service provisioning problem results in a GNEP, where the strategies of SaaS j are $r_j = (r_{ki})_{k\in\mathcal{A}_j, i\in\mathcal{I}_j}$, $d_j = (d_{ki})_{k\in\mathcal{A}_j, i\in\mathcal{I}_j}$, $s_j = (s_{ki})_{k\in\mathcal{A}_j, i\in\mathcal{I}_j}$ and $x_j = (x_{ki})_{k\in\mathcal{A}_j, i\in\mathcal{I}_j}$, while the strategies of IaaS i are $\sigma_i = (\sigma_{ji})_{j\in\mathcal{S}_i}$.

In this setting, a Generalized Nash Equilibrium (GNE) is a set of strategies such that no player can improve its payoff function by changing its strategy unilaterally, i.e., it is a vector $(r^*, d^*, s^*, x^*, \sigma^*)$ such that constraints (2)–(9) and (11)–(12) are satisfied, for any $j \in \mathcal{S}$ we have

$$\Theta_j(r_j^*, d_j^*, s_j^*, x_j^*, \sigma^*) \leq \Theta_j(r_j, d_j, s_j, x_j, \sigma^*) \tag{13}$$

for any (r_j, d_j, s_j, x_j) satisfying constraints (2)–(9), and for all $i \in \mathcal{I}$ we have

$$\Theta_i(r^*, d^*, s^*, x^*, \sigma_i^*) \geq \Theta_i(r^*, d^*, s^*, x^*, \sigma_i) \tag{14}$$

for any σ_i satisfying constraints (11)–(12).

5 Game analysis

In this section, we study the properties of the game formulated in Section 4.

Proposition 5.1. *For every IaaS $i \in \mathcal{I}$, the strategy $\sigma_{ji} = \sigma_{ji}^{Max}$, for any $j \in \mathcal{S}_i$, is a dominant strategy.*

Proof: Since the objective function Θ_i is non-decreasing with respect to σ_{ji} and each variable σ_{ji} belongs to the interval $[\omega_i, \sigma_{ji}^{Max}]$, it is clear that the maximum possible revenue is obtained setting $\sigma_{ji} = \sigma_{ji}^{Max}$. □

Hereafter, we suppose that each IaaS provider plays its dominant strategy. Therefore, IaaS providers can be dropped in the game formulation.

Proposition 5.2. *If $(r_j^*, d_j^*, s_j^*, x_j^*)$ is an optimal solution of SaaS j, then all the constraints (2) are active, i.e.*

$$D_{ki} + \frac{1}{\mu_{ki} - \frac{x_{ki}^*}{r_{ki}^* + d_{ki}^* + s_{ki}^*}} = \overline{R}_k, \quad \forall\, k \in \mathcal{A}_j,\ \forall\, i \in \mathcal{I}_j. \tag{15}$$

Proof: If, by contradiction, there exist $k \in \mathcal{A}_j$ and $i \in \mathcal{I}_j$ such that

$$D_{ki} + \frac{1}{\mu_{ki} - \frac{x_{ki}^*}{r_{ki}^* + d_{ki}^* + s_{ki}^*}} < \overline{R}_k,$$

then we could decrease the value of variables $r_{ki}^*, d_{ki}^*, s_{ki}^*$, if they are positive, of a sufficiently small quantity such that all the constraints remain satisfied. However, the new value of the cost function Θ_j would be less than the optimal value, which is impossible. □

Proposition 5.2 allows us to write constraints (2) as equality constraints, i.e.

$$x_{ki} = (\mu_{ki} - p_{ki})\,(r_{ki} + d_{ki} + s_{ki}), \qquad \forall\, k \in \mathcal{A}_j, \forall\, i \in \mathcal{I}_j, \tag{16}$$

where $p_{ki} := 1/(\overline{R}_k - D_{ki})$. Furthermore, it is easy to check that constraints (16) imply that constraints (3) are always fulfilled. Thanks to the previous results, the SaaS j problem can be reformulated as follows:

$$\begin{aligned} \min_{r_j, d_j, s_j, x_j} \Theta_j = \quad & \sum_{k\in\mathcal{A}_j} \sum_{i\in\mathcal{I}_j} [\rho_i\, r_{ki} + \delta_i\, d_{ki} + \sigma_{ji}^{Max}\, s_{ki} + \\ & - T\, \nu_k\, x_{ki}] + \sum_{k\in\mathcal{A}_j} T\, \nu_k\, \Lambda_k \end{aligned} \tag{17}$$

subject to:

$$\sum_{j\in\mathcal{S}_i}\sum_{k\in\mathcal{A}_j}(r_{ki}+d_{ki}+s_{ki})\le N_i,\ \ \forall\, i\in\mathcal{I}_j, \tag{18}$$

$$s_{ki}\le\frac{\eta_j}{1-\eta_j}\,(r_{ki}+d_{ki})\,,\ \ \forall\, k\in\mathcal{A}_j,\forall\, i\in\mathcal{I}_j, \tag{19}$$

$$\sum_{k\in\mathcal{A}_j} r_{ki}\le R_{ji},\ \ \forall\, i\in\mathcal{I}_j, \tag{20}$$

$$x_{ki}=(\mu_{ki}-p_{ki})\,(r_{ki}+d_{ki}+s_{ki})\,,\ \ \forall\, k\in\mathcal{A}_j,\forall\, i\in\mathcal{I}_j, \tag{21}$$

$$\lambda_k\le\sum_{i\in\mathcal{I}_j} x_{ki},\ \ \forall\, k\in\mathcal{A}_j, \tag{22}$$

$$\sum_{i\in\mathcal{I}_j} x_{ki}\le\Lambda_k,\ \ \forall\, k\in\mathcal{A}_j, \tag{23}$$

$$r_{ki},d_{ki},s_{ki},x_{ki}\ge 0,\ \ \forall\, k\in\mathcal{A}_j,\forall\, i\in\mathcal{I}_j. \tag{24}$$

We note that the SaaS optimization problem (17)–(24) is a linear programming problem, where constraints (18) are shared with the other SaaS providers.

The resulting equilibrium model is a GNEP with joint constraints, where the players are the SaaS providers only. Since the objective function (17) of each SaaS is independent of the strategies of all other SaaSs, we can conclude that this game is a Generalized Potential Game [30], where the potential function is the sum of player cost functions Θ_j.

Proposition 5.3. *The function*

$$\begin{aligned}\Pi\,(r,d,s,x)\quad =&\sum_{j\in\mathcal{S}}\sum_{k\in\mathcal{A}_j}\sum_{i\in\mathcal{I}_j}\Big[\rho_i\,r_{ki}+\delta_i\,d_{ki}+\sigma_{ji}^{Max}\,s_{ki}\\&-T\,\nu_k\,x_{ki}]+\sum_{j\in\mathcal{S}}\sum_{k\in\mathcal{A}_j}T\,\nu_k\,\Lambda_k\end{aligned}$$

is a potential for the game.

This means that the equilibria of our game coincide with those of a game where all the cost functions are equal to the potential Π. If we denote by $y_j:=(r_j,d_j,s_j,x_j)$ the strategy vector of SaaS j, by $y=(y_j)_{j\in\mathcal{S}}$ the strategy vector of all the players, and by Y the set of vectors y satisfying constraints (18)–(24) for all the SaaS providers, then the minimizers of Π over Y are called *social equilibria* of the game. It is clear from the above definitions that each social equilibrium is a special GNE; indeed, no single player can improve its payoff by unilaterally deviating from 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.

Proposition 5.4. *There exists at least one social equilibrium of the game.*

Proof: It is easy to check that the polyhedron Y is closed and bounded. Since the potential Π is a linear function on y, the existence of a social equilibrium follows directly from the Weierstrass Theorem. □

Ultimately, we want to remark that, being Π a linear function and Y a polyhedron, modern LP solvers can efficiently find a social equilibrium even for problems with several thousands of variables. In preliminary experiments we were able to solve instances of up to 10 IaaS, 1,000 SaaS and 10,000 WSs in less than 30 minutes.

Algorithm 1:

1 $\mathcal{I}_L=\emptyset$ // Initialization
2 **for** $j\in\mathcal{S}$ **do**
// Solution of relaxed subproblems
3 Find an optimal solution $\widetilde{y}_j$ of $\begin{cases}\min\ \Theta_j(y_j)\\ g_j(y_j)\le 0\end{cases}$
4 **repeat**
5 $\mathcal{I}_N=\{i\in\mathcal{I}\setminus\mathcal{I}_L : h_i(\widetilde{y})>0\}$
6 $N_{ij}=\dfrac{\sum_{k\in\mathcal{A}_j}(\widetilde{r}_{ki}+\widetilde{d}_{ki}+\widetilde{s}_{ki})}{\sum_{j\in\mathcal{S}_i}\sum_{k\in\mathcal{A}_j}(\widetilde{r}_{ki}+\widetilde{d}_{ki}+\widetilde{s}_{ki})}\,N_i \qquad \forall\, i\in\mathcal{I}_N,\ \forall\, j\in\mathcal{S}_i$
7 $\mathcal{I}_L=\mathcal{I}_L\cup\mathcal{I}_N$
8 **for** $j\in\mathcal{S}$ **do**
// Solution of new subproblems
9 Find an optimal solution $\widetilde{y}_j$ of
10 $\begin{cases}\min\ \Theta_j(y_j)\\ g_j(y_j)\le 0\\ h_{ij}(y_j)=\sum_{k\in\mathcal{A}_j}(r_{ki}+d_{ki}+s_{ki})-N_{ij}=0,\\ \qquad\qquad\qquad\qquad \forall\, i\in\mathcal{I}_L\cap\mathcal{I}_j\end{cases}$
11 **until** $\mathcal{I}_N=\emptyset$

6 A DISTRIBUTED ALGORITHM FOR IDENTIFYING GENERALIZED NASH EQUILIBRIA

From the theoretical point of view, a GNE could be computed by solving the social problem $\min_{y\in Y}\Pi(y)$, which is a Linear Programming problem. However, this is not possible in practice because it would require to share the information on the SLA contracts and performance parameters (i.e., ν_k, μ_{ki}, p_{ki}, etc.) among SaaS providers.

In order to overcome this drawback, we propose an *ad-hoc* distributed algorithm (see Algorithm 1) for finding a GNE. Roughly speaking, first each SaaS solves its own optimization problem without the joint constraints. If the optimal solutions satisfy the shared constraints, then a social equilibrium is found; otherwise, for each IaaS corresponding to a violated shared constraint, the VMs are shared proportionally among SaaS providers. Next, each SaaS solves its own relaxed problem with additional constraints: if all the shared constraints are satisfied in the new solution, then it is a GNE, otherwise we repeat the previous procedure for each violated constraint, and so on.

We now give a more detailed description of Algorithm 1. For the sake of simplicity, we denote constraints (19)–(24) of the SaaS j problem, which are independent of the strategies of other players, by $g_j(y_j)\le 0$ and constraints (18), shared with the other players, by

$$h_i(y):=\sum_{j\in\mathcal{S}_i}\sum_{k\in\mathcal{A}_j}(r_{ki}+d_{ki}+s_{ki})-N_i\le 0,\quad\forall\, i\in\mathcal{I}.$$

The algorithm performs the following steps. At the first iteration (lines 2-3), each SaaS j finds the optimal solution $\widetilde{y}_j$ of its own relaxed problem, in which the joint constraints are removed. As stated previously, if the solution $\widetilde{y}$ satisfies the shared constraints, then it is a social equilibrium. Otherwise, for each IaaS i such that the corresponding shared constraint h_i is violated by $\widetilde{y}$, the VMs of IaaS i are shared proportionally among SaaS providers according to the solution $\widetilde{y}$ (line 6), i.e., we define

$$N_{ij}:=\frac{\sum_{k\in\mathcal{A}_j}(\widetilde{r}_{ki}+\widetilde{d}_{ki}+\widetilde{s}_{ki})}{\sum_{j\in\mathcal{S}_i}\sum_{k\in\mathcal{A}_j}(\widetilde{r}_{ki}+\widetilde{d}_{ki}+\widetilde{s}_{ki})}\,N_i,\qquad\forall\, j\in\mathcal{S}_i,$$

which represents the number of resources of IaaS i that will be used by SaaS j in the successive iterations. Next, each SaaS provider solves its own problem with these new individual constraints (lines 9-10). If all the shared constraints are satisfied in the new solution, then it is a GNE, otherwise we compute N_{ij} for each violated constraint again and we add to each SaaS provider the corresponding constraints as before.

Note that Algorithm 1 can be implemented in a fully distributed manner: the SaaS providers initially send to the IaaSs their bid σ_{ji}^{Max} and the required value for the on-spot resources, i.e., s_{ki} obtained solving the relaxed subproblem at line 3. Then iteratively, the IaaS providers send back to individual SaaSs the effective on-spot cost and the number of VMs available N_{ij}. In the following, each SaaS provider solves independently its individual subproblem at lines 9-10. When Algorithm 1 terminates, each SaaS starts reserved, on-demand, and on-spot VMs according to the determined solution.

Theorem 6.1. *Algorithm 1 finds a Generalized Nash Equilibrium after a finite number of iterations.*

Proof: The set $\mathcal{I}_L$ represents the set of IaaS providers with limited resources with respect to the requests of SaaS providers at previous iterations, while $\mathcal{I}_N$ represents the set of IaaS providers with limited resources with respect to the requests of SaaS providers at the current iteration. Since $\mathcal{I}_L$ is a subset of $\mathcal{I}$ with increasing cardinality and $\mathcal{I}_N$ is a subset of $\mathcal{I} \setminus \mathcal{I}_L$, after a finite number of iterations we get $\mathcal{I}_N = \emptyset$, i.e., the algorithm stops.

Two cases can occur: either the algorithms stops at the first iteration with $\mathcal{I}_L = \emptyset$ or it stops after several iterations with $\mathcal{I}_L \neq \emptyset$.

In the first case, we prove that the found solution $\widetilde{y}$ is a social equilibrium. In fact, each SaaS provider j finds individually the optimal solution $\widetilde{y}_j$ of the relaxed problem without the shared constraints, thus there are optimal KKT multipliers $\widetilde{\beta}_j$ such that the following system holds for all $j \in \mathcal{S}$:

$$\begin{cases} \nabla\Theta_j(\widetilde{y}_j) + \widetilde{\beta}_j^T \nabla g_j(\widetilde{y}_s) = 0 \\ \widetilde{\beta}_j^T g_j(\widetilde{y}_j) = 0 \\ \widetilde{\beta}_j \geq 0 \\ g_j(\widetilde{y}_j) \leq 0. \end{cases}$$

Since $\widetilde{y}$ also satisfies all the shared constraints, i.e., $h_i(\widetilde{y}) \leq 0$ for all $i \in \mathcal{I}$, then $(\widetilde{y}, \widetilde{\beta}, \widetilde{\gamma})$, with $\widetilde{\gamma}_{ij} = 0$ for all $i \in \mathcal{I}$ and $j \in \mathcal{S}_i$, solves the KKT system associated to the social problem

$$\begin{cases} \min \sum_{j \in \mathcal{S}} \Theta_j(y_j) \\ g_j(y_j) \leq 0, \qquad \forall\, j \in \mathcal{S}, \\ h_i(y), \leq 0, \qquad \forall\, i \in \mathcal{I}. \end{cases}$$

Since the social problem is a linear programming problem, it follows that $\widetilde{y}$ is a social equilibrium.

We now consider the second case, i.e., the algorithm stops after several iterations with $\mathcal{I}_L \neq \emptyset$. We know that, for each SaaS j, $\widetilde{y}_j$ is an optimal solution of the optimization problem at line 10, thus there exist KKT multipliers $(\widetilde{\beta}_j, \widetilde{\gamma}_{ij})$ such that the following system holds:

$$\begin{cases} \nabla\Theta_j(\widetilde{y}_j) + \widetilde{\beta}_j^T \nabla g_j(\widetilde{y}_j) + \sum_{i \in \mathcal{I}_L \cap \mathcal{I}_j} \widetilde{\gamma}_{ij} \nabla h_{ij}(\widetilde{y}_j) = 0, \\ \widetilde{\beta}_j^T g_j(\widetilde{y}_j) = 0, \\ h_{ij}(\widetilde{y}_j) \quad = 0, \qquad \forall\, i \in \mathcal{I}_L \cap \mathcal{I}_j, \\ \widetilde{\beta}_j \quad \geq 0, \\ g_j(\widetilde{y}_j) \quad \leq 0. \end{cases}$$

We now prove that multipliers $\widetilde{\gamma}_{ij}$ are non-negative. In fact, if there was a $\widetilde{\gamma}_{ij} < 0$, then the SaaS j could reduce its objective function if the constraint on the VMs of the IaaS i was $\sum_{k \in A_j}(r_{ki} + d_{ki} + s_{ki}) < N_{ij}$, but this is impossible because in a previous iteration the SaaS j had already requested the IaaS i more than N_{ij} VMs.

For each $i \in \mathcal{I}_L$ we have $h_{ij}(\widetilde{y}_j) = 0$ for any $j \in \mathcal{S}_i$, which is equivalent to $\sum_{k \in \mathcal{A}_j}(\widetilde{r}_{ki} + \widetilde{d}_{ki} + \widetilde{s}_{ki}) = N_{ij}$, $\forall\, j \in \mathcal{S}_i$, thus

$$h_i(\widetilde{y}) = \sum_{j \in \mathcal{S}_i} \sum_{k \in A_j} (\widetilde{r}_{ki} + \widetilde{d}_{ki} + \widetilde{s}_{ki}) - N_i = \sum_{j \in \mathcal{S}_i} N_{ij} - N_i = 0,$$

i.e., the shared constraint h_i is active at $\widetilde{y}$.

Furthermore, we note that $\nabla h_{ij}(y_j) = \nabla_{y_j} h_i(y)$. If we define $\widetilde{\gamma}_{ij} = 0$ for all $i \in (\mathcal{I} \setminus \mathcal{I}_L) \cap \mathcal{I}_j$, then $\widetilde{y}_j$ satisfies the following system:

$$\begin{cases} \nabla\Theta_j(\widetilde{y}_j) + \widetilde{\beta}_j^T \nabla g_j(\widetilde{y}_j) + \sum_{i \in \mathcal{I}_j} \widetilde{\gamma}_{ij} \nabla_{y_j} h_i(\widetilde{y}) = 0, \\ \widetilde{\beta}_j^T g_j(\widetilde{y}_j) = 0, \\ \widetilde{\gamma}_{ij}\, h_i(\widetilde{y}) \quad = 0, \qquad \forall\, i \in \mathcal{I}_j, \\ \widetilde{\beta}_j \quad \geq 0, \\ g_j(\widetilde{y}_j) \quad \leq 0, \\ \widetilde{\gamma}_{ij} \quad \geq 0, \qquad \forall\, i \in \mathcal{I}_j, \\ h_i(\widetilde{y}) \quad \leq 0, \qquad \forall\, i \in \mathcal{I}_j. \end{cases}$$

This means that $\widetilde{y}_j$ is the optimal solution of the problem:

$$\begin{cases} \min\ \Theta_j(y_j) \\ g_j(y_j) \leq 0 \\ h_i(y_j, \widetilde{y}_{-j}) \leq 0, \qquad \forall\, i \in \mathcal{I}_j, \end{cases}$$

that is $\widetilde{y}_j$ is the best reply of player j to the strategies $\widetilde{y}_{-j}$ of the other players. Therefore, $\widetilde{y}$ is a Generalized Nash Equilibrium. □

We remark that the algorithm complexity is polynomial since it finds a GNE after resolving at most $|\mathcal{S}|(1+|\mathcal{I}|)$ linear programming problems.

7 Experimental results

In this section, we present some numerical results we achieved with the algorithm presented in Section 6 in a number of system configurations. The analysis is structured as follows: Section 7.1 describes the experiment settings, in Section 7.2 the scalability of the proposed approach is evaluated, Section 7.3 presents a comparative study on equilibria efficiency between our solution approach and an alternative state-of-the-art algorithm, Section 7.4 is devoted to analyze potential benefits that are achievable by a SaaS hosting its applications on multiple IaaSs. Finally, the impact of data replication on SaaS savings is analyzed in Section 7.5.

Note that, the comparison of different solutions is performed through analytical formulae and it is based on the evaluation of the response time through M/G/1 in tandem with delay center performance model. The validation of the approach in a real prototype environment has been reported in [16].

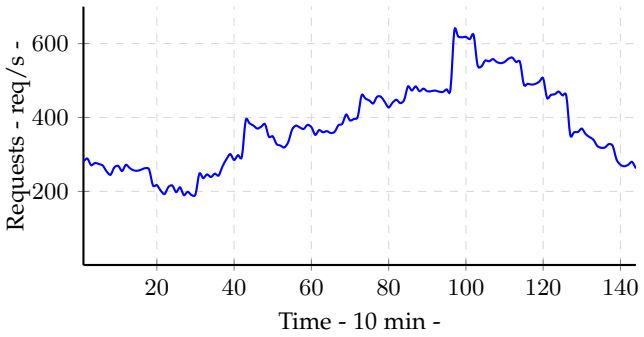

Figure 3. Example of WS application request trace.

7.1 Design of experiments

Analyses performed in this section are intended to be representative of a real Cloud environment. A very large set of randomly generated instances has been considered. Application performance parameters have been generated uniformly according to the ranges reported in Table 2, as in other literature approaches [4], [11], [17], [42]. Time unit costs have also been randomly generated by considering commercial fares applied by real IaaS Cloud providers [8], [36], [48].

μ_{ki}	$[200, 400]$ req/s	σ_{ji}^{Max}	$[0.013, 0.047]$ \$/h
$\overline{R}_k$	$[0.025, 0.100]$ s	ρ_i	$[0.048, 0.076]$ \$/h
D_{ki}	$[0.001, 0.050]$ s	δ_i	$[0.120, 0.175]$ \$/h

Table 2
Performance parameters and time unit costs.

For what concerns the distribution of incoming requests to WS applications, we used real-life measurements coming from a large website that, for privacy reasons, wants to remain anonymous. A 12-days log trace is sampled every 10 minutes. To differentiate workload of multiple WS applications, we applied to each application a different scaling factor and added a white noise to each sample, as in [11], [42]. The workload is periodic and follows a bimodal distribution with two peaks around 11:00 and 16:00.

Furthermore, with the aim of generating realistic scenarios in which applications receive requests worldwide distributed over multiple time zones, we considered Facebook annual report statistics [58] to characterize geographically requests distribution. Each application is then localized on a random basis. According to the assigned location, the daily pattern of arrival rate is adjusted to the correct time zone. In this way, WS applications can have multiple peak hours or overlapping peaks during the same day. An example of trace is reported in Figure 3. In order to limit request rejection, we always set $\lambda_k = 0.8\,\Lambda_k$.

7.2 Scalability analysis

To evaluate the computation and network overhead of our resource allocation algorithm, we performed a scalability analysis and considered a set of 100 randomly generated instances according to the parameters setting described above. The number of SaaS providers varied between 100 and 1,000, the number of applications (evenly shared among SaaSs) between 1,000 and 10,000. The values reported in this section are averaged over 10 instances of the same size. In order to guarantee the existence of competition among SaaSs, for each instance we calculate beforehand the optimal solution of the social problem with infinite capacity ($N_i = \infty,\ \ \forall i \in \mathcal{I}$) for every hour $(r_{ki}^t, d_{ki}^t, s_{ki}^t)$ and impose $N_i = 0.9 \times \max_{t \in T}(\sum_{k \in \mathcal{A}_j}(r_{ki}^t + d_{ki}^t + s_{ki}^t))\ \forall i \in \mathcal{I}$. We then assess the scalability of the presented approach by performing tests on a Virtualbox VM based on Ubuntu 14.04.1 LTS server running on an Intel Xeon Nehalem dual socket quad-core system with 32 GB of RAM and CPLEX 12.2.

Table 3 reports in the Appendix the results (in terms of average computation and network time, number of iterations, and bandwidth requirements of our solution) for different problem instance sizes, considering $|\mathcal{I}| = 10$ and $|\mathcal{I}_j| = 3$ for every $j \in \mathcal{S}$.

As a first consideration, we can state that, considering the computation time, our algorithm scales linearly with respect to the size of the instance remaining below 1.5s; as evidence, the coefficient of determination R^2 for the linear regression calculated over the collected data is equal to 0.931, showing a strong linear correlation between computation time and instance size. The computational time assumes that at each iteration SaaS problems are solved in parallel (i.e., a distributed implementation is adopted), while the network time considers the time required to access current on spot price and to set the price of running on spot instances on Amazon EC2 (we used the average time measured performing almost 200 tests through the Amazon EC2 python API [9] during 22-hour experiment).

As regards the network time, which is comparable with the computation time, it is proportional to the number of iterations and is almost constant, ranging between 1.19s and 1.34s. Table 3 reports also the peak bandwidth requirements for SaaSs and IaaSs. SaaS bandwidth is negligible in the order of kB/s. The same observation holds for IaaSs. In the very worst case, IaaS peak bandwidth is around 19 MB/s, which is affordable for a Cloud data center. The overall **`Algorithm 1`** total execution time is on average lower than 3s.

Finally, under the assumption to perform runtime resource allocation periodically on a hourly basis [4], [18], [19], we can assert that the proposed method is certainly efficient and it can be used at runtime, as it demonstrate to solve the largest problem instances in the very worst case in less than one minute.

7.3 Equilibria efficiency

This section is devoted to analyze the quality of our solution in the frame of an extensive experimental campaign in which also an alternative algorithm is considered. The next two sub-sections present a reference solution from the literature and a sound comparative evaluation based on established performance indicators.

7.3.1 Alternative algorithm

`Algorithm 2` is intended to mimic the typical policy, based on CPU utilization thresholds, implemented by SaaSs to manage incoming workload variations. In particular, the method is designed to calculate the number of VMs for a given WS application k, such that the average CPU utilization is lower than a given threshold $\overline{U}$. The algorithm can be

Algorithm 2:

Set $\mathcal{I}_L = \emptyset$
for $j \in \mathcal{S}$ **do**
  Find an optimal solution $\widetilde{y}_j$ of
  $\begin{cases} \min \Theta_j(y_j) \\ \widetilde{g}_j(y_j) \quad \texttt{// Initialization} \\ u_j(y_j) \leq \overline{U} \end{cases}$
$\mathcal{S}_N = \mathcal{S}$
repeat
  for $j \in \mathcal{S}$ **do**
    Find an optimal solution $\bar{y}_j$ of
    $\begin{cases} \min \Theta_j(y_j) \\ \widetilde{g}_j(y_j) \quad \texttt{// Best Reply} \\ u_j(y_j) \leq \overline{U} \end{cases}$
    if $\bar{y}_j \neq \widetilde{y}_j$ **then**
      $\widetilde{y}_j = \bar{y}_j$
      $\mathcal{S}_N = \mathcal{S}$
    else
      $\mathcal{S}_N = \mathcal{S}_N \setminus \{j\}$
until $\mathcal{S}_N = \emptyset$

described as a sequence of two steps executed for each SaaS provider: an initialization step performed to generate a first VM allocation, followed by a best reply mechanism aimed at finding a GNE of the game where each SaaS problem is defined by (26).

The crux of both steps is a typical policy based on CPU utilization thresholds, which leverages the auto-scaling mechanisms offered by IaaS providers (see, e.g., [7]) and considered also by other literature approaches (see, e.g., [25], [67], [71]). Therefore, we replaced in the problem defined by (1)–(9) the constraints (2)–(3), related on the response time, with a new one predicating on the utilization, i.e.:

$$u_j(y_j) = \frac{x_{ki}}{\mu_{ki}(r_{ki} + d_{ki} + s_{ki})} \leq \overline{U}, \qquad \forall\, k \in \mathcal{A}_j, \forall\, i \in \mathcal{I}_j. \quad (25)$$

Consequently, SaaS problem to solve becomes:

$$\begin{cases} \min \Theta_j(y_j) \\ \widetilde{g}_j(y_j) \leq 0 \\ u_j(y_j) \leq \overline{U}, \end{cases} \quad (26)$$

where $\widetilde{g}_j(y_j)$ denotes constraints (4)–(9). As in [25], [67], [71], we set $\overline{U} = 60\%$.

Notice that it is possible to demonstrate (in a similar fashion as done for Proposition 5.2) that constraints (25) are active in any optimal solution of problem (26).

Finally, since $\widetilde{g}_j(y_j)$ relies on the knowledge of strategies implemented by the other players of the game (see constraint (6)), **`Algorithm 2`** must be implemented as a centralized solution.

7.3.2 Comparative analysis

In the following, we denote with $(\widetilde{r}, \widetilde{d}, \widetilde{s}, \widetilde{x})$ a social optimum solution and with $(\overline{r}, \overline{d}, \overline{s}, \overline{x})$ a solution found by **`Algorithm 1`**. Under this assumption, the equilibria efficiency can be measured in terms of *Price of Anarchy* (PoA) and of *Indivual Worst Case* (IWC), defined as:

$$\text{PoA} = \frac{\Pi(\overline{r}, \overline{d}, \overline{s}, \overline{x})}{\Pi(\widetilde{r}, \widetilde{d}, \widetilde{s}, \widetilde{x})}, \qquad \text{IWC} = \max_{j \in \mathcal{S}} \frac{\Theta_j(\overline{r}, \overline{d}, \overline{s}, \overline{x})}{\Theta_j(\widetilde{r}, \widetilde{d}, \widetilde{s}, \widetilde{x})}.$$

PoA and IWC represent a measure of the inefficiency due to IaaSs and SaaSs selfish behavior with the 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 every single SaaS provider. Moreover, we analyze how equilibria efficiency varies with respect to the percentage ϕ_i of reserved instances at IaaS i, i.e., $\phi_i := \left(\sum_{j \in \mathcal{S}_i} R_{ji} \right) / N_i$. Withal, since data regarding IaaSs trade policies and resource overbooking are not public, we consider three values of ϕ_i, namely 10%, 30% and 50%.

Furthermore, with the purpose of comparing the algorithms in situations in which resource contention increases, we gradually decreased the value of N_i for each IaaS, until no feasible allocation is found. With a high value of N_i a small fraction of VMs available at each IaaS are used by SaaS providers and consequently the shared constraints are not active. The service provisioning problem becomes easy to solve since there is no competition among players and each SaaS problem can be considered separately and solved to optimality (note that in such cases PoA is equal to 1). Vice versa, by reducing N_i, the service provisioning problem becomes more complex, as shared constraints are active and SaaS players compete for resources assignment.

Results obtained for $|\mathcal{I}| = 10, |\mathcal{S}| = 100, |\mathcal{A}| = 1,000$, and $|\mathcal{I}_j| = 3$ for any $j \in \mathcal{S}$ are summarized in the Appendix by Table 4.

`Algorithm 1` shows a very good behavior since all the solutions found are very close to the social optimum. The PoA is lower than 1.03 (i.e., on average, the percentage difference of the sum of the payoff functions with respect to the social optimum is lower than 3%), while the IWC is lower than 1.37 (i.e., in the worst case, the cost of a SaaS provider in the social equilibrium is 37% greater than in the GNE). Finally, the analysis of the ratio $\boldsymbol{X_k/\Lambda_k}$ tells us that when the problem addressed become very difficult to solve, because only few VMs are available, SaaSs are willing to reject a small fraction of the input load to keep the response time below the threshold $\overline{R}_k$.

As regards **`Algorithm 2`**, we do not present the values of PoA and IWC, as we did for our algorithm, because having replaced conditions (2) with utilization constraints (25) we radically changed the structure of the problem and they have less or no meaning in such condition. Instead, we decided to calculate and report the averaged values of ratio $\boldsymbol{\Pi_{\texttt{Algorithm 2}}/\Pi_{\texttt{Algorithm 1}}}$ (that is the ratio between the total cost of solution returned by **`Algorithm 2`** and by **`Algorithm 1`** for the same instance) and the total number of request violations associated to the equilibrium found. As the reader can see, the outcomes of **`Algorithm 2`** are about 82%, more expensive on average. Furthermore, the algorithm identifies equilibria with greater difficulty, failing to converge to feasible solutions from $N_i = 70/65$ onwards and introducing also a significant number of response time violations (evaluated as the total number of requests such that $E[R_k] > \overline{R}_k$). With this respect an increasing trend with ϕ_i is also evident, meaning that with the growing resource contention the algorithm tends to return worse quality solutions in terms of SLA. Finally, it is evident that the utilization threshold based policies commonly implemented are far from being optimal, being slower (doubling the number of iterations to converge), more expensive (∼55% more VMs allocated) and failing in guaranteeing the SLAs (some

WS applications are over-provisioned, others are under-provisioned).

7.4 Multiple IaaS analysis

This section is devoted to the study of pros, cons and possible trade-offs arising when SaaSs rely on more than one IaaS to host their applications. In this regard, we performed a campaign of experiments considering SaaS providers leasing resources from one up to three IaaSs at the same time and we compared the results in terms of efficiency and total execution cost (i.e., payoff function value).

In order to perform a fair comparison with the single IaaS case, the number of reserved resources is kept constant and independent of the number of IaaSs available, that is:

$$\sum_{i \in \mathcal{I}_j,\, |\mathcal{I}_j|=1} R_{ji} = \sum_{i \in \mathcal{I}_j,\, |\mathcal{I}_j|=2} R_{ji} = \sum_{i \in \mathcal{I}_j,\, |\mathcal{I}_j|=3} R_{ji}\,, \qquad \forall\, j \in \mathcal{S}.$$

The performance index under study is the average value of the payoff function (PF), which represent the SaaS total daily cost net of replication costs and, like in Section 7.3, we consider three values for $\phi_i \in \{0.1,\, 0.3,\, 0.5\}$. The payoff function, therefore, catches only the cost associated to the use of resources and penalties (in \$); the impact of data replication and synchronization is analyzed separately in the next section.

Let us start the analysis considering a Cloud environment characterized by $\phi_i = 0.1$, which means that each single IaaS offers a maximum of 10% of its virtual machines as reserved. A summary of average results is reported by Table 5 in the Appendix, where the value of the PF is expressed in \$.

It appears evident that having more than one IaaS to exploit is an advantage for the SaaS if we consider only the payoff function and the reliability. With the use of two IaaS providers, there are, indeed, savings of 9.83% with respect to the single-IaaS deployment, and up to 15.26% when the SaaS leverages three IaaSs. This result is due to the fact that the only IaaS involved in the experiment offers a limited number of reserved VMs and the SaaS is forced to run the on-demand ones, more expensive; this choice, however, is often better than paying the penalties for unsatisfied SLAs.

The same trend can be identified in the results of the experiments performed with $\phi_i = 0.3$ and $\phi_i = 0.5$ whose outcomes are also collected and summarized in Table 5. There is still a 8.95% and 13.59% (10.60% and 15.16%, respectively) savings on average when deploying applications on two and three IaaSs.

After discussing in deep the results of this experiment, we can ultimately draw some considerations on the behavior of a SaaS provider, which we have obtained by inspecting equilibrium solutions. A SaaS provider, according to the workload conditions, can be in one of the following four situations:

1) The workload is satisfied only leasing reserved and, proportionally to the η_j parameter, spot instances at the cheapest IaaS provider. In this way the SaaS can easily guarantee that all requests will be served and the pledged QoS will be granted to its customers.

However, if restrictions in resource allocations exist, due, for instance, to a limited number of resources available at the IaaS side, or by the competition of other SaaS players, or to the low amount of reserved instances (e.g., for a low ϕ_i), it has to enact adequate countermeasures:

2) In the case the SaaS profits from using more than one IaaS, it can ask a different IaaS to provision the extra resources needed to satisfy the incoming traffic or bidding for the cheapest spot resources on multiple sites, with the aim of exploiting the lowest price zones. In doing so, the SaaS provider is also improving the reliability of its applications portfolio.
3) Differently, if the SaaS cannot rely on multiple IaaSs or if the SaaS has already consumed the reserved instances and the on spot ratio has been reached, it has no other choice but to use the most expensive type of VMs, the on demand ones.
4) Withal, there exists a forth, albeit unlikely, scenario in which a SaaS could decide to reject part of the incoming workload and pay the resulting penalties. In the experimental analysis we set the cost associated with request rejection much higher (> 10 times) than the execution cost of the most expensive VMs in the game, in order to make the rejection an extreme and expensive choice. Nonetheless, there are some conditions that impose the SaaS to reject part of the workload; it happens when the total number of VMs is unable to cover the resource requirements of all SaaS providers. In this case the SaaS, behaving according to the model described in Section 4.1, rejects some requests to keep the response time under the thresholds $\overline{R}_k$.

To conclude, the analysis presented in this section clearly demonstrates that simultaneously exploiting more than one IaaS can be really beneficial for the SaaS from a monetary and reliability point of view. In fact, in terms of profits, a SaaS would achieve savings ranging from a minimum of 10% up to 15%. Finally, the advantages related to the reliability, which can be guaranteed to end users, are also considerable. As a matter of fact, the reliability of the system increases with the number of used IaaSs, raising, for instance, from 99.9% (i.e., around 6 hours of downtime per year) to 99.9999% (11 sec/year) when two IaaSs are considered together (according to the Cloud providers reliability measures reported in [26]).

7.5 Impact of Data replication on Multi-IaaS cost savings

We now evaluate the impact of data replication and synchronization on multi-IaaS cost savings reported and analyzed in Section 7.4. We do this by means of a sample application (Meeting in the Cloud: MiC). The outcomes of this experiments are collected and analyzed with the goal of assessing the viability of multi-Cloud solutions also from the monetary point of view. The research question we try to answer is: provided a suitable multi-Cloud data synchronization system, how strong is the impact of the inter-Cloud data transfer on the SaaS execution costs?

In the following, first we briefly introduce the MiC application and then we present the experiment carried on and analyze the results.

7.5.1 Case study: Meeting in the Cloud (MiC) application

To validate our approach and evaluate the cost overhead introduced by data synchronization, we have implemented a sample application, which leverages several data storage services [35]. MiC is a social networking web application. It allows a user to register and to choose his/her topics of interest providing a grade in the range 1-5. At the end of the registration process, MiC identifies the most similar users in the social network according to the user's preferences [44], [45] (similarity is computed through the *Pearson coefficient* [52]). Then, the user can access to the portal and interact with his/her *Best Contacts* creating and reading posts on the selected topics.

The application is developed in Java using JSP and Servlet technologies and the following data storage services are used:

- Blob Service: used to store profile pictures.
- NoSQL Service: used for storing user interests and preferences.
- SQL Service: used for storing user profiles, messages, and best contacts.

7.5.2 Cost analysis of Data replication impact

To study the impact of data replication in terms of supplementary cost incurred to maintain the consistency in a multi-IaaS deployment, we have to calculate the amount of data that has to be synchronized among multiple Clouds. In this sense, if ξ_k is the average size of records written by WS application k, the data transfer rate from IaaS i_1 to IaaS i_2 for each WS application k can be calculated as $z_{k\,i_1\,i_2} = \alpha_k\, x_{k\,i_1}\, \xi_k$ for any $k \in \mathcal{A}_j$ and $i_1, i_2 \in \mathcal{I}_j$, where $\alpha_k \geq 0$ is a parameter that characterizes the I/O behavior of the application. Higher values of α_k describe an application that tends to execute several write operations for every incoming request, whereas lower values are representative of an application prone to execute read operations.

If WS application k is deployed on $|\mathcal{I}_j|$ different Clouds, to achieve full data replication $z_{k\,i_1\,i_2} = z_{k\,i_1}$ bytes per second must be transferred from i_1 to each of the $|\mathcal{I}_j| - 1$ IaaSs. Therefore, the amount of data that have to be sent from Cloud i during a certain period of time T is

$$\Xi_{ki} = (|\mathcal{I}_j| - 1)\, T z_{ki}, \tag{27}$$

whereas the cost associated to the synchronization process can be calculated as

$$C_{Synch} = \sum_{i \in \mathcal{I}_j} \mathcal{C}^i_{out}(\Xi_{ki}), \tag{28}$$

where $\mathcal{C}^i_{out}$ is a function representing the fee applied by the IaaS provider i for the transfer of data.

At the time of writing, all the major Cloud providers do not apply fees for inbound data flows (that is the SaaS is not charged for data coming from outside the Cloud) but they charge egress costs that depends on the traffic spawned (in TB) per month. Moreover the cost functions are usually concave due to the economy of scale policies adopted by the IaaS providers.

For our study we considered $|\mathcal{S}| = 100, |\mathcal{A}| = 100$, $\phi_i \in \{0.1, 0.3, 0.5\}$ and varied the number of involved IaaSs $|\mathcal{I}| = |\mathcal{I}_j|$ in [1, 3].

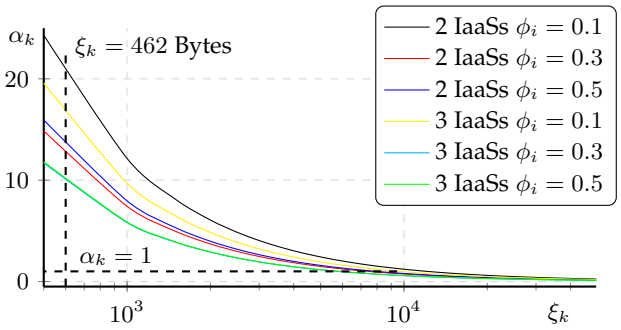

Figure 4. Break-even curves for $\phi_i \in \{0.1, 0.3, 0.5\}$ and $|\mathcal{I}| \in \{2, 3\}$

We use our solution to calculate the total execution cost and the related incoming workload split z_{ki} for each application and each IaaS for a time horizon of 24 hours.

At this point, to calculate the average egress cost we need two important pieces of information: the values of α_k and ξ_k. For our reference MiC application, the average data record ξ_k is known and equal to 462 bytes, while α_k depends on end users behaviour (e.g., how many posts are written per visit to the social network). To determine a general insight of our problem, we performed a parametric break-even analysis, that is the study of the values of α_k corresponding to the balance point between the multi-IaaS cost savings and the data synchronization cost and their dependence on ξ_k. The results of the analysis are depicted in Figure 4.

For this study we use the pricing models of three real IaaS providers, namely Amazon AWS [8], Google GCP [37] and Microsoft Azure [47].

As the reader can easily note, for all the experiments conducted the curves of α_k show high values (ranging from 12 to 24) for $\xi_k = 462$ bytes meaning that, depending on the case, the application can execute up to 12 (24, respectively) writing operations per request. Moreover, the break-even point for $\alpha_k = 1$ (that describes the scenario where there is one writing operation for each incoming request) corresponds to a data record of ~10 MB, which proves that our approach is viable even for data intensive applications or with a significant overhead added by the synchronization tool.

As a final remark: in principle, the synchronization costs (28) might be introduced in the game formulation presented in Section 4 but this would eventually only complicate the model (making it non-linear) without changing much neither the social optima nor the equilibria found by our algorithm. In fact, for values of $\alpha_k < 1$ (more realistic for Web applications [17], [61]) we have seen that synchronization has little impact on the overall execution cost and it is even less relevant if we consider that all major Cloud providers are applying very similar fees for the egress traffic and thereby it cannot influence the choice of a IaaS provider against another.

8 Conclusions

The overall goal addressed in this paper is to model and efficiently solve the cost minimization problem associated with the allocation of SaaS virtual machines in multiple IaaSs, while guaranteeing QoS constraints. To achieve this purpose, we proposed a game-theoretic approach for the

runtime management of IaaSs provider capacities among multiple competing SaaSs along with a cost model including infrastructural costs associated with IaaSs resources and penalties incurred for request rejections. A distributed algorithm for identifying a Generalized Nash Equilibrium has been developed and described in detail. The effectiveness of our approach is demonstrated by performing a wide set of analyses under multiple workload conditions. A number of different scenarios of interest have been considered. Systems up to thousands of applications can be managed very efficiently in a fully distributed way. Our algorithm found efficient GNE, for a hourly basis resource allocation, in less than a minute proving to be perfectly suitable for runtime provisioning. Furthermore, a comparison with an utilization-based state-of-the-art technique demonstrated that our solution outperforms the alternative method in many aspects and shows high equilibrium efficiency, with both PoA and IWC close to one. Finally, analyses showed clear benefits for SaaSs that decide to exploit multiple IaaS deployment and traffic redistribution with average savings up to 15% compared to single IaaS architectures.

Future work will extend the proposed solution to consider multiple time-scales ranging from few minutes to one hour, exploiting also receding horizon techniques along the lines we developed in [13] and we will consider also different IaaS on spot pricing scheme.

ACKNOWLEDGEMENT

This work is partially supported by the European Commission grant no. FP7-ICT-2011-8-318484 (MODAClouds).

REFERENCES

[1] V. Abhishek, I. A. Kash, and P. Key. Fixed and market pricing for cloud services. In *2012 Proceedings IEEE INFOCOM Workshops, Orlando, FL, USA, March 25-30, 2012*, pages 157–162, 2012.

[2] M. Abundo, V. Di Valerio, V. Cardellini, and F. Presti. Bidding Strategies in QoS-Aware Cloud Systems Based on N-Armed Bandit Problems. In *NCCA*, 2014.

[3] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, and L. Zhang. A hierarchical approach for the resource management of very large cloud platforms. *IEEE Trans. on Dependable Sec. Comput.*, 10(5):253–272, 2013.

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

[5] E. Altman, U. Ayesta, and B. J. Prabhu. Optimal load balancing in processor sharing systems. In *GameComm*, pages 12:1 –12:10, 2008.

[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 autoscaling. http://aws.amazon.com/autoscaling/.

[8] Amazon Inc. Amazon EC2 pricing. http://aws.amazon.com/ec2/pricing/.

[9] Amazon Inc. Aws sdk for python (boto). http://aws.amazon.com/sdk-for-python/.

[10] J. Anselmi and B. Gaujal. Optimal routing in parallel, non-observable queues and the price of anarchy revisited. In *ITC*, 2010.

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

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

[13] D. Ardagna, M. Ciavotta, and R. Lancellotti. A receding horizon approach for the runtime management of iaas cloud systems. In *MICAS-SYNASC*, 2014.

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

[15] D. Ardagna, B. Panicucci, and M. Passacantando. A game theoretic formulation of the service provisioning problem in cloud systems. In *WWW*, 2011.

[16] D. Ardagna, B. Panicucci, and M. Passacantando. Generalized Nash equilibria for the service provisioning problem in cloud systems. *IEEE Trans. on Services Computing*, 6(4):429–442, 2013.

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

[18] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, 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, 2009.

[19] R. Birke, L. Chen, and E. Smirni. Data centers in the cloud: A large scale performance study. In *CLOUD*, 2012.

[20] 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*, pages 235–263, 2003.

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

[22] S. Casolari and M. Colajanni. Short-term prediction models for server management in internet-based contexts. *Decision Support Systems*, 48(1), 2009.

[23] E. Cavazzuti, M. Pappalardo, and M. Passacantando. Nash equilibria, variational inequalities, and dynamical systems. *Journal of Optimization Theory and Applications*, 114(3):491–506, 2002.

[24] 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, 2008.

[25] 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):669–685, 2002.

[26] CloudHarmony Inc. Cloudsquare service status. https://cloudharmony.com/status-1year-of-compute-group-provider.

[27] G. Debreu. A social equilibrium existence theorem. *Proceedings of the National Academy of Sciences of the USA*, 38(10):886–893, 1952.

[28] V. Di Valerio, V. Cardellini, and F. Lo Presti. Optimal pricing and service provisioning strategies in cloud systems: A Stackelberg game approach. In *CLOUD*, 2013.

[29] F. Facchinei and C. Kanzow. Generalized Nash Equilibrium Problems. *Annals OR*, 175(1):177–211, 2010.

[30] F. Facchinei, V. Piccialli, and M. Sciandrone. Decomposition algorithms for generalized potential games. *Computational Optimization and Applications*, pages 237–262, 2011.

[31] Y. Feng, B. Li, and B. Li. Price competition in an oligopoly market with multiple IaaS cloud providers. *IEEE Trans. on Computers*, 63(1):59–73, 2014.

[32] O. Flinker. Microsoft Azure suffers series of multi-region cloud outages. https://www.cloudendure.com/blog/microsoft-azure-suffers-series-multi-region-cloud-outages/.

[33] D. Gamarnik, Y. Lu, and M. S. Squillante. Fundamentals of stochastic modeling and analysis for self-* properties in autonomic computing systems. Technical report, IBM Research, October 2004.

[34] D. Gamarnik, Y. Lu, and M. S. Squillante. Workload management based on the heavy-traffic theory of queueing systems. Technical report, IBM Research, September 2004.

[35] F. Giove, D. Longoni, M. Shokrolahi, D. Ardagna, and E. D. Nitto. An approach for the development of portable applications on paas clouds. In *CLOSER*, 2013.

[36] Google Inc. Compute engine. https://cloud.google.com/compute/.

[37] Google Inc. Google cloud platform pricing calculator. https://cloud.google.com/products/calculator.

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

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

[40] M. Jebalia, A. Ben Letaïfa, M. Hamdi, and S. Tabbane. A comparative study on game theoretic approaches for resource allocation in cloud computing architectures. In *WETICE*, 2013.

[41] D. Kumar, L. Zhang, and A. N. Tantawi. Enhanced inferencing: estimation of a workload dependent performance model. In *VALUETOOLS*, pages 47:1 – 47:10, 2009.

[42] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[43] J. Künsemöller and H. Karl. A game-theoretic approach to the financial benefits of infrastructure-as-a-service. *Future Generation Computer Systems*, 41(0):44–52, 2014.

[44] R. Lancellotti, S. Casolari, and C. Canali. A quantitative methodology to identify relevant users in social networks. In *BASNA*, 2010.

[45] T.-P. Liang, H.-J. Lai, and Y.-C. Ku. Personalized content recommendation and user satisfaction: Theoretical synthesis and empirical findings. *J. of Management Information Systems*, 23(3):45–70, 2007.

[46] I. Menache, A. Ozdaglar, and N. Shimkin. Socially optimal pricing of cloud computing resources. In *VALIETOOLS*, 2011.

[47] Microsoft. Microsoft azure. http://azure.microsoft.com/en-us/pricing/details/data-transfers/.

[48] Microsoft. Microsoft azure pricing. http://azure.microsoft.com/en-us/pricing/calculator/?scenario=virtual-machines.

[49] MODAClouds. http://www.modaclouds.eu/.

[50] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[51] R. Pal and P. Hui. Economic models for cloud service markets: Pricing and capacity planning. *Theoretical Computer Science*, 496(0):113–124, 2013.

[52] K. Pearson. Similarity Metrics: Pearson Correlation Coefficient. http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/sphilip/pear.html.

[53] A. S. Prasad and S. Rao. A mechanism design approach to resource procurement in cloud computing. *IEEE Trans. on Computers*, 63(1):17–30, 2014.

[54] A. Riska, M. Squillante, S. Z. Yu, Z. Liu, and L. Zhang. *Matrix-Analytic Analysis of a MAP/PH/1 Queue Fitted to Web Server Data. In Matrix-Analytic Methods: Theory and Applications, G. Latouche and P. Taylor Ed.*, pages 333 – 356. World Scientific, 2002.

[55] H. Roh, C. Jung, W. Lee, and D.-Z. Du. Resource pricing game in geo-distributed clouds. In *INFOCOM*, 2013.

[56] J. B. Rosen. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica*, 33(3):520–534, 1965.

[57] M. Scavuzzo, E. D. Nitto, and D. Ardagna. Building Data Intensive Applications Exploiting Data as a Service Systems: Experiences and Challenges. Politecnico di Milano Technical report n. 2014.10. http://home.deib.polimi.it/ardagna/DataReplication2014-10.pdf.

[58] Socialbakers. Facebook statistics. http://www.socialbakers.com/facebook-statistics/.

[59] Y. Song, S. H. Wong, and K.-W. Lee. Optimal gateway selection in multi-domain wireless networks: A potential game perspective. In *MobiCom*, 2011.

[60] Y. Song, M. Zafer, and K.-W. Lee. Optimal bidding in spot instance market. In *INFOCOM*, 2012.

[61] SPEC. Specweb 2009 benchmark. https://www.spec.org/web2009, 2009.

[62] F. Teng and F. Magoules. A new game theoretical resource allocation algorithm for cloud computing. In *GPC*, 2010.

[63] C.-W. Tsai and Z. Tsai. Bid-proportional auction for resource allocation in capacity-constrained clouds. In *WAINA*, 2012.

[64] V. Vinothina, R. Sridaran, and P. Ganapathi. A Survey on Resource Allocation Strategies in Cloud Computing. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 3(6):97 – 104, 2012.

[65] W. Wang, B. Li, and B. Liang. Towards optimal capacity segmentation with hybrid cloud pricing. In *ICDCS*, 2012.

[66] Y. Wang, X. Lin, and M. Pedram. A Game Theoretic Framework of SLA-Based Resource Allocation for Competitive Cloud Service Providers. In *GreenTech*, 2014.

[67] A. Wolke and G. Meixner. TwoSpot: A Cloud Platform for Scaling Out Web Applications Dynamically. In *ServiceWave*, 2010.

[68] B. Yolken and N. Bambos. Game based capacity allocation for utility computing environments. In *VALUETOOLS*, pages 8:1 – 8:8, 2008.

[69] L. Zhang, X. Meng, S. Meng, and J. Tan. K-scope: Online performance tracking for dynamic cloud applications. In *ICAC*, pages 29–32, 2013.

[70] Q. Zhang, Q. Zhu, M. Zhani, and R. Boutaba. Dynamic service placement in geographically distributed clouds. In *ICDCS*, 2012.

[71] 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.

D anilo Ardagna is an Associate Professor at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano. He received a Ph.D. degree in computer engineering in 2004 from Politecnico di Milano, from which he also graduated in December 2000. His work focuses on the design, prototype and evaluation of optimization algorithms for resource management of self-adaptive and cloud systems.

M ichele Ciavotta received the Ph.D. degree in automation and computer science from Roma Tre, Italy in 2008. From 2012 he is Postdoctoral Fellow at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano. His research work focus on modeling and optimization of complex real-life problems mainly arising in the fields scheduling and planning, and more recently resource management for cloud-based and data intensive systems under constraints of quality of service.

M auro Passacantando received the Laurea and the PhD degree in mathematics from the University of Pisa, Italy, in 2000 and 2005, respectively. He is currently an assistant professor of Operations Research in the Department of Computer Science, University of Pisa. He has authored several scientific papers in international journals and books. His research is mainly devoted to variational inequalities and equilibrium problems, concerning both theory and algorithms, and their applications to game theory.

APPENDIX

($\lvert\mathcal{S}\rvert$, $\lvert\mathcal{A}\rvert$)	Iter.	Comput. Time (s)	Network Time (s)	IaaS Peak BW (MB/s)	SaaS Peak BW (kB/s)	Tot. Exec. Time (s)
(100, 1000)	1.81	0.226	1.23	1.83	56.15	1.46
(200, 2000)	1.91	0.388	1.30	3.86	59.26	1.69
(300, 3000)	1.96	0.549	1.34	5.94	60.81	1.89
(400, 4000)	1.93	0.686	1.31	7.78	59.77	2.00
(500, 5000)	1.86	0.792	1.27	9.39	57.70	2.06
(600, 6000)	1.83	0.915	1.24	11.07	56.67	2.16
(700, 7000)	1.74	0.993	1.19	12.32	54.08	2.18
(800, 8000)	1.88	1.152	1.28	15.16	58.22	2.43
(900, 9000)	1.78	0.964	1.22	16.22	55.37	2.18
(1000, 10000)	1.85	1.307	1.26	18.70	57.45	2.57

Table 3
Execution times of the distributed algorithm for identifying GNE.

		Algorithm 1					Algorithm 2				
ϕ_i	N_i	PoA	IWC	Iter.	$\frac{X_k}{\Lambda_k}$	VMs	$\frac{\Pi_{\text{Algorithm 2}}}{\Pi_{\text{Algorithm 1}}}$	Iter.	Viol.	$\frac{X_k}{\Lambda_k}$	VMs
0.1	100	1.0005	1.0032	1.00	1.00	324.57	1.76	2.00	98407.2	1.00	508.54
	95	1.0005	1.0032	1.00	1.00	324.48	1.75	2.00	86762.2	1.00	508.50
	90	1.0005	1.0032	1.00	1.00	324.40	1.74	2.00	90405.4	1.00	508.46
	85	1.0005	1.0032	1.00	1.00	324.33	1.73	2.00	84021.7	1.00	508.43
	80	1.0005	1.0032	1.00	1.00	324.26	1.71	2.00	76317.7	1.00	508.41
	75	1.0005	1.0032	1.00	1.00	324.20	1.70	2.00	70879.6	1.00	508.46
	70	1.0005	1.0032	1.00	1.00	324.15	1.69	2.00	63016.8	1.00	508.70
	65	1.0005	1.0031	1.00	1.00	324.09	-	-	-	-	-
	60	1.0005	1.0031	1.00	0.99	324.05	-	-	-	-	-
	55	1.0005	1.0031	1.00	0.99	324.01	-	-	-	-	-
	50	1.0005	1.0032	1.05	0.98	323.98	-	-	-	-	-
	45	1.0007	1.0040	1.29	0.96	324.01	-	-	-	-	-
	40	1.0066	1.1010	1.85	0.93	322.51	-	-	-	-	-
	35	1.0261	1.3617	2.21	0.93	307.63	-	-	-	-	-
0.3	100	1.0010	1.0083	1.00	1.00	327.04	1.96	2.00	168449.0	1.00	512.09
	95	1.0009	1.0078	1.00	1.00	327.22	1.98	2.00	173619.1	1.00	511.77
	90	1.0008	1.0074	1.00	1.00	327.35	1.99	2.00	188790.6	1.00	511.46
	85	1.0007	1.0069	1.00	1.00	327.40	2.00	2.00	191636.9	1.00	511.15
	80	1.0007	1.0065	1.00	1.00	327.34	2.00	2.00	193170.1	1.00	510.84
	75	1.0006	1.0060	1.00	1.00	327.17	1.99	2.00	195921.0	1.00	510.58
	70	1.0006	1.0055	1.00	1.00	326.92	1.97	2.00	182617.5	1.00	510.46
	65	1.0006	1.0051	1.00	1.00	326.59	-	-	-	-	-
	60	1.0005	1.0047	1.00	0.99	326.25	-	-	-	-	-
	55	1.0005	1.0043	1.00	0.98	325.92	-	-	-	-	-
	50	1.0005	1.0039	1.03	0.96	325.59	-	-	-	-	-
	45	1.0007	1.0045	1.23	0.95	325.31	-	-	-	-	-
	40	1.0073	1.1187	1.83	0.93	323.46	-	-	-	-	-
	35	1.0223	1.3434	2.18	0.91	308.18	-	-	-	-	-
0.5	100	1.0023	1.0121	1.00	1.00	324.38	1.64	2.00	29458.2	1.00	512.49
	95	1.0022	1.0119	1.00	1.00	324.57	1.66	2.00	35201.0	1.00	512.81
	90	1.0021	1.0116	1.00	1.00	324.83	1.69	2.00	53262.0	1.00	513.09
	85	1.0019	1.0112	1.00	1.00	325.13	1.73	2.00	71002.8	1.00	513.30
	80	1.0017	1.0107	1.00	1.00	325.50	1.77	2.00	80637.3	1.00	513.38
	75	1.0016	1.0102	1.00	1.00	325.90	1.82	2.00	110850.9	1.00	513.29
	70	1.0014	1.0096	1.00	1.00	326.31	1.87	2.00	123348.9	1.00	513.09
	65	1.0012	1.0090	1.00	1.00	326.69	1.92	2.00	145239.0	1.00	509.35
	60	1.0010	1.0083	1.00	0.99	327.04	-	-	-	-	-
	55	1.0008	1.0075	1.00	0.96	327.32	-	-	-	-	-
	50	1.0007	1.0068	1.00	0.95	327.40	-	-	-	-	-
	45	1.0007	1.0069	1.14	0.93	327.18	-	-	-	-	-
	40	1.0082	1.1486	1.71	0.90	324.91	-	-	-	-	-
	35	1.0177	1.3091	1.94	0.86	309.06	-	-	-	-	-

Table 4
Approach comparison results

	PF		
$\lvert\mathcal{I}_j\rvert$	$\phi_i = 0.1$	$\phi_i = 0.3$	$\phi_i = 0.5$
1	78.20	52.58	47.53
2	70.50	47.88	42.49
3	66.26	45.44	40.33

Table 5
Multi-IaaS analysis results