Optimal Pricing and Service Provisioning Strategies in Cloud Systems: A Stackelberg Game Approach[*]

Valerio Di Valerio Valeria Cardellini Francesco Lo Presti
Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università di Roma "Tor Vergata", Roma, Italy
di.valerio@ing.uniroma2.it, cardellini@ing.uniroma2.it, lopresti@info.uniroma2.it

Università di Roma "Tor Vergata"
Dipartimento di Ingegneria Civile e Ingegneria Informatica
Technical Report RR-13.01

February 20, 2013

Abstract

Cloud computing is a recent advancement where service providers offer hardware infrastructures, platforms, and software applications as services to users on the basis of a usage-based pricing model. As Cloud-based systems have been increasing in size and complexity and operate in a continuously and unpredictably changing environment, it has become increasingly more important and challenging to develop effective provisioning and pricing schemes.

In this paper we consider several Software as a Service (SaaS) providers, that offer a set of Web services using the Cloud facilities provided by an Infrastructure as a Service (IaaS) provider. We assume that the IaaS provider offers a *pay only what you use* scheme similar to the Amazon EC2 service, comprising flat, on demand, and spot virtual machine instances. We propose a two stage provisioning scheme. In the first stage, the SaaS providers determine the number of required flat and on demand instances by means of standard optimization techniques. In the second stage the SaaS providers compete, by bidding for the spot instances which are instantiated using the unused IaaS capacity. We assume that the SaaS providers want to maximize a suitable utility function which accounts for both the QoS delivered to their users and the associated cost. The IaaS provider, on the other hand, wants to maximize his revenue by determining the spot prices given the SaaS bids. We model the second stage as a Stackelberg game, and we compute its equilibrium price and allocation strategy by solving a Mathematical Program with Equilibrium Constraints (MPEC) problem. Through numerical evaluation we study the equilibrium solutions as function of the system parameters.

[*]This Technical Report has been issued as a Research Report for early dissemination of its contents. No part of its text nor any illustration can be reproduced without written permission of the Authors.

1 Introduction

Cloud computing has recently been experiencing a high rate of grow, mainly due to the ability of realizing highly scalable and reliable infrastructures for running software applications in an efficient and cost-effective way. Industrial companies and research communities have started using commercially available infrastructures provided by Infrastructure as a Service (IaaS) providers to run their applications, that can scale up or down as demand changes by allocating or deallocating virtual computing and storage resources almost instantaneously. In their turn, customers of IaaS providers can rapidly offer their innovative applications, thus becoming Software as a Service (SaaS) providers, but without needing to own and maintain development or production infrastructures.

Among the different methods to deliver Cloud services, an IaaS provider can sell resources in the form of Virtual Machine (VM) instances to customers, that generally rent the resources by using a pay-as-you-go model on a per-hour basis at a fixed price (also called *on demand* price). When resource utilization can be planned in advance, IaaS customers can also reserve *flat* resources in advance, paying a long-term reservation fee plus a per-hour price which depends on the effective resource usage and is lower than the on demand price. In order to achieve high utilization in data centers that are often under-utilized, IaaS providers can also sell their spare capacity in form of *spot* instances by organizing an auction where customers bid, providing a maximum per-hour price they are willing to pay. On the basis of the bids and his spare capacity, the IaaS provider sets the spot instances price. For IaaS customers spot instances represent an attractive and cost-effective solution to deal with unexpected load spikes and run compute-intensive applications but at the risk of a lower reliability than flat and on demand instances, since the IaaS provider can revoke spot instances without notice due to price and demand fluctuations [18, 24]. For example, Amazon's Elastic Cloud Computing (EC2) service offers three types types of VM instances (i.e., flat, on demand, and spot VMs) [3], with different pricing and reliability.

From an IaaS perspective, selling resources to multiple customers requires to determine efficient service provisioning and pricing strategies, in order to maximize the IaaS resources usage and the provider profit and to satisfy the customers. Developing such strategies in the Cloud environment is a challenging task, mostly because the Cloud environment is inherently competitive and dynamic. On the one hand, the IaaS provider wants to maximize his profit and the resources usage; on the other hand, the customers compete among them to acquire the resources and are interested in saving money. In this context, game theoretic methods can help to analytically model and understand the service provisioning and pricing problem and to device adequate strategies. Game theory has been already successfully applied to networking problems like Internet congestion control and pricing, e.g., [2] and in the Cloud computing environment, e.g., [4, 5, 15, 23]. In most works, the solution concept of Nash equilibrium has been extensively applied (a set of players' strategies is a Nash equilibrium if no player one can improve his revenue by changing its strategy unilaterally, i.e., while the other players keep theirs unchanged).

In this paper, we consider a Cloud scenario where an IaaS provider sells his resources to several SaaS providers, offering flat, on demand, and spot VM instances. In their turn, SaaS providers offer to their end users Web services with Quality of Service (QoS) guarantees, using the IaaS facilities to host and run the provided applications. Revenues and penalties of each SaaS provider depend on the provisioning of an adequate performance level, which is specified in a Service Level Agreement (SLA) contract that each SaaS stipulates with his end users. Therefore, each SaaS provider has to face the problem of determining the optimal number of VMs to satisfy the SLA with his end users while maximizing his revenue. However, a proper solution cannot be accomplished in isolation, since the SaaS providers compete among them and bid to acquire the spot IaaS resources. On the other hand, the IaaS provider aims to maximize his revenue and therefore wants to properly choose the price of his resources. In other words, each player strategy influences what the other players do.

To model this conflicting situation we recur to a Stackelberg game [13]. In this class of games, one player (i.e., the leader, in our case the IaaS provider) moves first and commits his strategy to the remaining players (i.e., the followers, in our case the SaaS providers), that consider the action chosen by the leader before acting simultaneously to choose their own strategy in a selfish way through a standard Nash game. Stackelberg games are commonly used to model attacker-defender scenarios, e.g. [7, 17]. For the considered Cloud scenario, the adoption of a leader-follower strategy sounds feasible, since it is reasonable to assume that the IaaS provider fixes the price before the SaaS providers compete to acquire the VM resources. Furthermore, a Stackelberg game help us to devise a revenue-maximizing pricing scheme for the IaaS provider. Using game theory results, we develop an algorithm for the pricing and allocation of the IaaS resources that aims to maximize the IaaS provider profit and, at the same time, to satisfy the SaaS providers needs.

The main contributions of the present work are as follows.

- We devise a *two-stage* resource provisioning and pricing strategy. In the *first stage*, each SaaS provider independently determines the optimal number of flat and on demand VMs (which have a fixed price) in such a way to guarantee the performance level offered in the SLA to his end users while maximizing his profit. In the *second stage*, the SaaS providers bid and compete for the unused IaaS provider capacity. The goal of each SaaS provider is to determine the number of spot instances to allocate which maximizes his profit, given the number of flat and on demand instances bought in the first stage. The goal of the IaaS provider is to determine the price of the spot VMs in order to maximize his profit. While the first stage involves the solution of standard optimization problems, the second stage requires to compute the equilibria of the SaaS/IaaS Stackelberg game on spot instances.

- We address the solution of the challenging Stackelberg game by solving a suitable Mathematical Program with Equilibrium Constraint (MPEC) problem.

- We study through numerical investigation the behavior of the proposed provisioning and pricing strategy under different workload and bidding configurations and compare it with the service provisioning and pricing policy proposed in [4]. Using our proposed strategy, the IaaS provider can set a price of the spot instances lower than the maximum price in the bids in order to incentivize SaaS providers to buy more instances, thus increasing his revenue with a higher volume of sold instances.

The rest of this paper is organized as follows. In Section 2 we define the system model and provide the problem statement. In Section 3 we define the two-stage service provisioning and pricing strategy, first analyzing the flat and on demand virtual machines provisioning that occurs in the first stage and then focusing on the second stage during which the number and price of spot instances is determined. We discuss the solution method of the Stackelberg game that arises from our problem formulation in Section 4. In Section 5 we analyze through numerical experiments the behavior of the proposed strategy and compare its results to those of the Generalized Nash Equilibrium Problem (GNEP) formulation proposed in [4]. In Section 6 we discuss related works. Finally, in Section 7 we conclude the paper and present directions for future work.

2 System Model

We consider a set $\mathcal{U}$ of SaaS providers that offer a set of Web services/applications $\mathcal{A}_u$, $u \in \mathcal{U}$, using the cloud facilities offered by a IaaS provider. We assume that each service $k \in \mathcal{A}_u$ is characterized by a SLA which stipulates the service QoS levels, *i.e.*, service response time, and the associated cost/penalty for its use.

Web services are hosted on virtual machines instantiated by the IaaS provider. For the sake of simplicity, we assume that the IaaS provider offers only one type of VMs, *i.e.*, all the VMs have the same RAM and CPU capacity. Each VM hosts only one Web service; on the other hand, each Web service can be distributed on multiple VMs and in that case we assume the workload to be evenly split among them.

The IaaS provider manages an infrastructure which can provide to his users up to $\mathcal{S}$ VMs which are offered to users as flat, on demand and/or spot instances. Flat instances are characterized by one-time payment plus a payment of φ unit per hour of actual use. On demand instances have no one-time payment and are charged at a price δ, which we assume to be strictly larger than φ. On spot instances are charged at a price $\sigma_{u,k}$, which we assume it may vary from SaaS provider to SaaS provider and from application to application, and which depends on the users bids and competition for the unused resources and the IaaS provider optimal pricing strategy.

Given the number of flat $f_{u,k}$, on demand $d_{u,k}$, and spot instances $s_{u,k}$ and their prices $\sigma_{u,k}$ allocated to service $k \in \mathcal{A}_u$, $u \in \mathcal{U}$, the associated per-hour IaaS revenue is:

$$\Theta_I = \sum_{u \in \mathcal{U}} \sum_{k \in \mathcal{A}_k} \varphi f_{u,k} + \delta d_{u,k} + \sigma_{u,k} s_{u,k} \tag{1}$$

Each SaaS provider determines for each service k the number of flat $f_{u,k}$, on demand $d_{u,k}$, and spot $s_{u,k}$ VMs to be allocated which maximizes his revenue, given the predicted arrival rate $\Lambda_{u,k}$ and the service SLA.

In this paper, we assume that SLA takes the form of an upper bound on the service response time $R^{max}_{u,k}$. The SLA also specifies the user per-request cost $\mathcal{C}_{u,k} = C_{u,k}(1 - \frac{R_{u,k}}{R^{max}_{u,k}})$, which we assume to be a linear function of the service response time $R_{u,k}$. Observe that the service cost is a decreasing function of the response time and it becomes negative (hence, the SaaS provider incurs a penalty) when $R_{u,k} > R^{max}_{u,k}$. We adopt such a simple model since linear costs allow to implement a soft constraint on the response time, which enables the SaaS provider to trade-off revenues and infrastructural costs [5].

We model each Web service hosted on a VM as an M/G/1/PS queue with an application dependent service rate $\mu_{u,k}$. Under the assumption of perfect load sharing among multiple VMs assigned to the same service, the service k average response time is given by:

$$E[R_{u,k}] = \frac{f_{u,k} + d_{u,k} + s_{u,k}}{\mu_{u,k}(f_{u,k} + d_{u,k} + s_{u,k}) - \Lambda_{u,k}}$$

provided the stability condition $\frac{\Lambda_{u,k}}{\mu_{u,k}(f_{u,k}+d_{u,k}+s_{u,k})} < 1$ holds. Taking into account the infrastructural per hour cost for the allocated VMs we have the following general expression for the per-hour SaaS profit:

$$\begin{aligned}
&\Theta_u = \\
&\sum_{k \in \mathcal{A}_u} \Lambda_{u,k} C_k \left(1 - \frac{1}{R^{max}_{u,k}} \frac{f_{u,k} + d_{u,k} + s_{u,k}}{\mu_{u,k}(f_{u,k} + d_{u,k} + s_{u,k}) - \Lambda_{u,k}}\right) \\
&- \varphi \sum_{k \in \mathcal{A}_u} f_{u,k} - \delta \sum_{k \in \mathcal{A}_u} d_{u,k} - \sum_{k \in \mathcal{A}_u} \sigma_{u,k} s_{u,k}
\end{aligned} \tag{2}$$

where the first term is the sum of the average per service revenues $\Lambda_{u,k}\mathcal{C}_k = \Lambda_{u,k} C_k (1 - \frac{E[R_{u,k}]}{R^{max}_{u,k}})$ and the remaining terms the VMs costs.

For the sake of clarity, we summarize in Table 1 the notation adopted in this paper. Furthermore, throughout the rest of this paper we will use the variable $m_{u,k} = -\frac{C_{u,k}}{R_{u,k}^{max}}$ to indicate the slope of the SaaS profit function.

System parameters	
$\mathcal{S}$	Total amount of VMs managed by the IaaS provider
$\mathcal{U}$	Set of SaaS providers
N	Number of SaaS providers
$\mathcal{A}_u$	Set of services for SaaS provider u
f_u	Number of reserved flat instances for SaaS provider u
$\Lambda_{u,k}$	Predicted arrival rate for service k of SaaS provider u
$\mu_{u,k}$	Maximum service rate of a unitary capacity VM executing service k of SaaS provider u
$m_{u,k}$	Utility function slope for service k of SaaS provider u
U_u^{max}	Maximum allowed utilization of a VM executing services of SaaS provider u
φ	Time unit cost for one *flat* VM
δ	Time unit cost for one *on demand* VM
σ^L	Minimum time unit cost for one *spot* VM, set by the IaaS provider
$\sigma_{u,k}^U$	Maximum time unit cost for one *spot* VM used for service k, set by the SaaS provider

Decision variables	
$f_{u,k}$	Number of *flat* VMs used for service k of SaaS provider u
$d_{u,k}$	Number of *on demand* VMs used for service k of SaaS provider u
$s_{u,k}$	Number of *spot* VMs used for service k of SaaS provider u
$\sigma_{u,k}$	Time unit cost for *spot* VMs used for service k of SaaS provider u

Table 1: Main notation adopted in the paper

3 Service Provisioning: a Stackelberg Game Approach

We assume that SaaS providers every hour allocate and deallocate VMs relying on a prediction of the future workload. In this paper we consider a two-stage allocation procedure. In the first stage, each SaaS provider independently determines, for each offered service, the number of flat and on demand instances[1] which guarantee the performance level defined in the SLA to its prospective users and maximize his profit. In the second stage, the IaaS provider sells the unused capacity as spot instances. We assume that the SaaS providers compete for these additional resources by submitting, to the IaaS provider, a bid which defines the maximum per VM price their are willing to pay. The IaaS provider, given the resource availability and the submitted bids, determines the spot instances price which maximizes his profit.

3.1 First Stage: Flat and on Demand VM allocation

In the first stage, each SaaS provider independently determines the optimal number of flat and on demand VMs necessary to sustain the predicted load for the next hour which maximizes his profit. Here we make the implicit assumption that the IaaS provider always has enough resources to accomodate all flat and on demand instances the users may require (otherwise we would have competition also at this stage). For each SaaS provider $u \in \mathcal{U}$ we have the following optimization

[1]Observe that, in case of flat instances, this number represents the number of *already* allocated flat instances which will be used to implement the offered services (a SaaS provider does not pay the per hour cost of unused flat instances).

problem:

$$\textbf{max } \Theta_u \tag{3}$$

$$\textbf{subject to: } \sum_{k \in \mathcal{A}_u} f_{u,k} \leq f_u \tag{4}$$

$$\frac{\Lambda_{u,k}}{\mu_{u,k}(f_{u,k} + d_{u,k})} \leq U_u^{max}, \quad \forall k \in \mathcal{A}_u \tag{5}$$

$$f_{u,k}, d_{u,k} \geq 0, \quad \forall k \in \mathcal{A}_u \tag{6}$$

Constraint (4) ensures that the flat instances allocated to SaaS provider u are less than or equal to the number of reserved ones f_u. Constraint (5) guarantees that resources are not saturated. In particular, it ensures that the resources utilization is less than a threshold U_u^{max}. This term forces the system to work away from the saturation point. Note that, as in [4, 5], we do not impose to the variables to be integers as in reality they are, because the problem would have been much more difficult to solve (NP-hard). Therefore, throughtout the paper we deal with a relaxation of the real problem. We believe that, nonetheless, our findings apply to the actual problem as well.

3.2 Second Stage: Spot Instances Allocation

In the second stage, the SaaS providers compete for the unused IaaS provider resources made available via a bidding mechanism. The idea is that the SaaS providers can increase their revenues by accessing additional resources, while the IaaS provider can make profit from the otherwise unsold resources. We will assume that, nevertheless, the Iaas provider sets a minimum bid σ^L for spot VMs he is willing to accept.

The goal of each SaaS provider is to determine the number of spot instances to allocate which maximizes his profit given the number of flat and on demand instances bought in the first stage. In the second stage, we assume that each SaaS provider u specifies $\sigma_{u,k}^U$, the maximum time unit cost for spot VMs for application k he is willing to pay (see the IaaS problem), and solves the following optimization problem:

Problem SaaS$_{OPT}$

$$\textbf{max } \Theta_u = \textbf{max} \sum_{k \in \mathcal{A}_u} \frac{m_{u,k}\Lambda_{u,k}(\bar{f}_{u,k} + \bar{d}_{u,k} + s_{u,k})}{\mu_{u,k}(\bar{f}_{u,k} + \bar{d}_{u,k} + s_{u,k}) - \Lambda_{u,k}} - s_{u,k}\sigma_{u,k}$$

$$\textbf{subject to: } \sum_{u \in \mathcal{U}} \sum_{k \in \mathcal{A}_u} s_{u,k} \leq s^U \tag{7}$$

$$s_{u,k} \geq 0, \quad \forall k \in \mathcal{A}_u \tag{8}$$

where $\bar{f}_{u,k}$ and $\bar{d}_{u,k}$ represents the number of flat and on demand instances already bought and

$$s^U = \mathcal{S} - \sum_{u \in \mathcal{U}} \sum_{k \in \mathcal{A}_u} (\bar{f}_{u,k} + \bar{d}_{u,k})$$

the amount of unused IaaS capacity, being $\mathcal{S}$ the total amount of VMs the IaaS provider manages and $\sum_{u \in \mathcal{U}} \sum_{k \in \mathcal{A}_u} (\bar{f}_{u,k} + \bar{d}_{u,k})$ the amount of VMs instantiated after the first stage. In this optimization problem constraint (7) ensures that the total number of spot VMs allocated to the SaaS providers are less that or equal to the ones available at the IaaS provider. Note that differently from the flat and on demand provisioning problem the SaaS providers solve in the first stage, we now have a constraint which involves the decision variables of all the SaaS providers and which is parametrized by the spot instances price $\sigma_{u,k}$, which is a IaaS decision variable.

The goal of the IaaS provider is to determine the cost $\sigma_{u,k}$ of the spot VMs for each service k of every SaaS provider u in order to maximize his profit. The IaaS provider optimization problem is:

$$\begin{aligned} &\textbf{Problem IaaS}_{OPT} \\ &\textbf{max } \Theta_I = \textbf{max} \sum_{u \in \mathcal{U}} \sum_{k \in \mathcal{A}_u} s_{u,k} \sigma_{u,k} \\ &\textbf{subject to: } \sigma^L \leq \sigma_{u,k} \leq \sigma^U_{u,k}, \quad \forall u \in \mathcal{U},\ \forall k \in \mathcal{A}_u \end{aligned} \tag{9}$$

where $\sigma^U_{u,k}$ is the maximum price for spot instances each SaaS provider is willing to pay, while σ^L is imposed by the IaaS provider and represents the minimum spot instances cost which takes into account the operative cost to run a VM.

In this setting, the decisions of the SaaS providers and the IaaS provider depend mutually from each other. Indeed, the objective function of the IaaS provider depends on $s_{u,k}$, the decision variables of the SaaS providers, while the objective function of each SaaS provider depends on $\sigma_{u,k}$, the price of the spot instances, which are the decision variables of the IaaS provider. Moreover, the decision of each SaaS provider depends also on what the others providers do, since constraint (7) couples the variables of all the SaaS providers.

We model such a conflicting situation as a Stackelberg game [13]. Stackelberg games are a particular type of non-cooperative game whereby one player (the leader) takes its decision before the other players (the followers). Given the leader decision, the followers then simultaneously take their own decision. Assuming a rational behavior, the leader can take advantage of the fact that the followers basically react to its decisions, which leads to a follower *subgame* equilibrium (if any exists), and drives the system to its own optimum (possibly a global one). In our model, the IaaS provider acts as a leader by deciding the spot instances prices $\sigma_{u,k}$. The SaaS providers act as followers which must decide the number $s_{u,k}$ of spot instance to buy. Given the spot prices, the SaaS providers thus compete among themselves (the SaaS subgame) for the shared pool of available instanced s^U. Following the Nash equilibrium concept, given the spot prices, the SaaS providers should adopt a strategy that none of them can improve their revenue by unilaterally changing their strategy. In turn, the IaaS provider can determine the sets of prices which maximizes his revenue and which represents the Stackelberg equilibrium of the game: this equilibrium is characterized by the property that for the given set of prices, the SaaS providers have adopted a strategy such that none of them could improve his revenue by changing it unilaterally, and the IaaS provider would not benefit by modifying the chosen set of prices since no other SaaS providers equilibrium would improve his revenue.

Observe the difference between Nash and Stackelberg equilibrium. In the former case, at the equilibrium no player can improve his payoff by *unilaterally* changing his own strategy, *i.e.*, while the other players keep their strategy unchanged. In Stackelberg equilibrium, while each follower, as in Nash equilibrium, cannot improve his payoff by *unilaterally* changing his own strategy, the leader cannot improve his payoff by changing his own strategy which, due to the nature of the game, always affects the strategy of the followers and cannot be accounted for independently.

4 Solution Method

Our provisioning scheme comprises two stages. The first stage involves the solution of a set of independent convex optimization problems which can be solved by means of standard techniques.

The second stage requires the computation of the *equilibria* of the SaaS/IaaS spot instance Stackelberg game. This is a challenging problem for which no general solution exists. In this paper, we take advantage of the structure and properties of the SaaS provider subgame, and we compute the Stackelberg equilibria by solving a suitable *Mathematical Programs with Equilibrium Constraint* (MPEC). In this section, we first study the SaaS providers subgame and establish some important game properties. We will then present an algorithm to compute an equilibrium of the Stackelberg game.

Throughout the rest of the section, we will denote by $s_u = (s_{u,k})_{k \in A_u}$ the strategy of a SaaS provider $u \in \mathcal{U}$ and with $\sigma = (\sigma_{u,k})_{u=1}^{N}$ the strategy of the IaaS provider, where $N = |\mathcal{U}|$. Furthermore, we indicate with $s = (s_u)_{u=1}^{N}$ the set of strategies of all the SaaS providers, with s^{-u} the set of the strategies of all the SaaS providers except the SaaS provider u. For the sake of simplicity, we also rewrite the SaaS providers optimization problem $\mathbf{SaaS}_{OPT}$ in compact form as follows:

$$\begin{aligned} &\mathbf{max}\ \Theta_u(s;\sigma) \\ &\mathbf{subject\ to:}\ g_u(s) \leq 0 \end{aligned}$$

where $g_u(s)$ is a vector function that represents the problem constraints. We define with $K_u = \{s\ |g_u(s) \leq 0\}$ the SaaS provider u feasible strategies set and we further define

$$\Omega = \left\{ s \left| \sum_{u \in \mathcal{U}} \sum_{k \in \mathcal{A}_u} s_{u,k} \leq s^U \right. \right\}.$$

Hence, we can define the vector function $g(s) = (g_u(s))_{u=1}^{N}$ that represents the set

$$K = (K_1 \times K_2 \ldots K_N) \cap \Omega$$

of the feasible strategies of all the players. Note that the set K is compact, since each variable $s_{u,k}$ is bounded by s^U.

4.1 SaaS Providers Subgame

We first consider the followers subgame, i.e., the SaaS providers competition that arises once the IaaS provider fixes his strategy (the spot prices σ). Since the SaaS providers act simultaneously, this subgame can be modeled as a Generalized Nash game [11]. Generalized Nash Equilibrium Problems (GNEP) differ from Nash Equilibrium Problems (NEP), in that, while in a NEP problem only the players objective functions depend on the other players strategies, in a GNEP problem both the objective functions and the strategy sets depend on the other players strategies. In our problem, the dependence of each player strategy set on the other players strategies is represented by the constraint (7) which include all SaaS providers decision variables $\sigma_{u,k}$. More specifically, our problem is a *Jointly Convex* GNEP [11]. Convexity follows from concavity of the objective function of each SaaS provider in his own decision variable and the convexity of the strategy set; furthermore, it is jointly convex because the constraint involving all players variables is the same for all players.

Jointly Convex GNEP problems are a particular class GNEP whose solution can be computed by solving a proper *variational inequality* (VI)[2]. In particular, under the condition that the objective function of each player is continuously differentiable, every solution of the $VI(K, F(s;\sigma))$, where

[2]Given a subset K of $\Re^n$ and a function $F : K \to \Re^n$, the VI problem, denoted by $VI(K, F)$, consists in finding a point $s^* \in K$ such that $(s - s^*)^T F(x^*) \geq 0 \quad \forall s \in F$.

$F(s;\sigma) = -[(\nabla_{s_u}\Theta_u(s;\sigma)_{u=1}^N)$, is also an equilibrium of the GNEP [11]. Such equilibrium is known as *variational equilibrium.* In general, a GNEP has multiple or even infinite equilibria, and not all of them are also a solution of the VI. However, the variational equilibrium is more "socially stable" than the other equilibrium of a GNEP and therefore it represents a valuable target for an algorithm [11].

We now establish two key properties of the $VI(K, F(x;\sigma))$, namely that the function F is strongly monotone[3] and the existence of the generalized Nash equilibrium of the followers subgame.

Theorem 1. *The function* $F(s;\sigma) = -[(\nabla_{s_u}\Theta_u(s;\sigma)_{u=1}^N)$ *is strongly monotone.*

Proof. The function F takes the following form:

$$F(s;\sigma) = \begin{bmatrix} \dfrac{m_{1,1}\Lambda_{1,1}^2}{(\mu_{1,1}(\bar{f}_{1,1}+\bar{d}_{1,1}+s_{1,1})-\Lambda_{1,1})^2}+\sigma_{1,1} \\ \cdots \\ \cdots \\ \cdots \\ \dfrac{m_{N,k}\Lambda_{N,k}^2}{(\mu_{N,k}(\bar{f}_{N,k}+\bar{d}_{N,k}+s_{N,k})-\Lambda_{N,k})^2}+\sigma_{N,k} \end{bmatrix}$$

and its Jacobian is:

$$JF(s;\sigma) = \begin{bmatrix} a_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_{N,k} \end{bmatrix}$$

JF is a diagonal matrix, whose generic entry $a_{u,k}$ is:

$$a_{u,k} = -\frac{2m_{u,k}\Lambda_{u,k}^2\mu_{u,k}}{(\mu_{u,k}(\bar{f}_{u,k}+\bar{d}_{u,k}+s_{u,k})-\Lambda_{u,k})^3}.$$

$a_{u,k}$ is strictly positive, because in the first stage we impose that

$$\bar{f}_{u,k}+\bar{d}_{u,k} > \frac{\Lambda_{u,k}}{\mu_{u,k}}$$

and $m_{u,k} < 0$. The generic term $a_{u,k}$ attains its minimum when $s_{u,k} = s^U$. If we choose a constant $0 < \alpha < \min_{k\in\mathcal{A}_u, u\in\mathcal{U}} a_{u,k}$ then the matrix $JF(s;\sigma) - \alpha I$ is still diagonal with each term strictly greater than 0, for all $s \in K$. Hence the function $F(s;\sigma)$ is strongly monotone. □

Theorem 2. *There exists at least one generalized Nash equilibrium of the followers subgame.*

Proof. The existence of at least one generalized Nash equilibrium for the followers subgame is a direct consequence of the strong monotonicity of the function $F(s;\sigma)$ [12]. □

[3] F is monotone on K if for all pairs $x, y \in K$ holds

$$(s-y)^T(F(s)-F(y)) \geq 0$$

F is strictly monotone on K if for all pairs $s \neq y \in K$ holds

$$(s-y)^T(F(s)-F(y)) > 0$$

F is strongly monotone on K if there exists a constant $c > 0$ such that for all pairs $s, y \in K$ holds

$$(s-y)^T(F(s)-F(y)) > c\| s-y\|^2$$

4.2 Stackelberg Game as MPEC

In the previous subsection we have established that the problem of finding an equilibrium of the SaaS providers subgame given the IaaS strategy σ can be reformulated as the problem of finding the solution of a proper VI. We now turn our attention to the IaaS (leader) problem of determining the optimal pricing strategy. We solve the Stackelberg game using a *Mathematical Programs with Equilibrium Constraint* (MPEC) [10]. An MPEC is an optimization problem whose constraints include variational inequalities. In its general form, an MPEC is defined by the following optimization problem:

$$\begin{aligned} &\mathbf{max}\ f(\sigma, s) \\ \mathbf{subject\ to:}\ &\sigma \in S_{lead} &(10)\\ &s \in SOL(\sigma) &(11)\end{aligned}$$

where $SOL(\sigma)$ is the solution of a VI parametrized by the leader strategy σ. Because of constraint (11), an MPEC is, in general, a nonconvex and nonsmooth problem, even under very favourable assumptions, and, as a consequence, it is very difficult to solve. The MPEC arising from our IaaS optimization takes the following form:

Problem IaaS$_{MPEC}$

$$\begin{aligned} \mathbf{max}\ &\sum_{u\in\mathcal{U}}\sum_{k\in\mathcal{A}_u} \bar{f}_{u,k}\varphi + \bar{d}_{u,k}\delta + s_{u,k}\sigma_{u,k} \\ \mathbf{subject\ to:}\ &\sigma^L \le \sigma_{u,k} \le \sigma^U_{u,k}, \quad \forall u, \forall k &(12)\\ &s \in SOL(\sigma) &(13)\end{aligned}$$

where $SOL(\sigma)$ is the solution of the $VI(K, F(s;\sigma))$. Observe that this is just the IaaS problem **IaaS**$_{OPT}$ with the additional constraints that the SaaS providers strategy s is a solution of the SaaS subgame.

Because of (13) we cannot solve the MPEC directly. We follow the approach proposed in [10] that, under the assumption that function $F(s;\sigma)$ is strongly monotone, allows us to compute stationary points of the MPEC.

As a first step we replace (13) with its Karush Kuhn Tucker (KKT) conditions [19]. We obtain the following non linear programming problem:

$$\begin{aligned} \mathbf{max}\ &\sum_{u\in\mathcal{U}}\sum_{k\in\mathcal{A}_u} \bar{f}_{u,k}\varphi + \bar{d}_{u,k}\delta + s_{u,k}\sigma_{u,k} \\ \mathbf{subject\ to:}\ &\sigma^L \le \sigma_{u,k} \le \sigma^U_{u,k}, \quad \forall u\ \forall k &(14)\\ &F(\sigma) - \nabla_s g(s)\lambda = 0 &(15)\\ &g(s) \le 0 &(16)\\ &\lambda \ge 0 &(17)\\ &\lambda^T g(s) = 0 &(18)\end{aligned}$$

where $\lambda \in \Re^l$ is the vector of Lagrangian multiplier, with l the number of constraints that define K. This problem cannot be directly solved because in general the constraints do not satisfy any standard constraints qualification and the complementary-type constraints (18) are very complicated and difficult to handle. Following [10], we consider a sequence of smooth and regular problems, obtained by perturbing the original problem, the solutions of which converge to a solution of the original problem. Specifically, we consider the pertubated problem $\mathbf{P}(\mu)$ with parameter μ:

Problem $\mathbf{P}(\mu)$

$$\textbf{max} \sum_{u \in \mathcal{U}} \sum_{k \in \mathcal{A}_u} \bar{f}_{u,k}\varphi + \bar{d}_{u,k}\delta + s_{u,k}\sigma_{u,k}$$

$$\textbf{subject to: } \sigma^L \leq \sigma_{u,k} \leq \sigma^U_{u,k},\ \forall u\ \forall k \tag{19}$$

$$F(\sigma) - \nabla_s g(s)\lambda = 0 \tag{20}$$

$$g(s) + z = 0 \tag{21}$$

$$\sqrt{(z-\lambda)^2 + 4\mu^2} - (z + \lambda) = 0 \tag{22}$$

where $z \in \Re^l$ is an auxiliary variable. In $\mathbf{P}(\mu)$, the constraint (21) replace constraint (16), while constraint (22) replaces constraints (17) and (18). It is easy to realize that $\mathbf{P}(\mu)$ corresponds to the original problem when $\mu = 0$[4]. We refer the reader to [10] for further details.

Problem $\mathbf{P}(\mu)$, $\mu \neq 0$, is a smooth regular problem which can be solved using standard optimization tools. Let $\sigma^*(\mu)$ denote a stationary point of $\mathbf{P}(\mu)$. From [10], we have that $\sigma^*(\mu)$ converges to a stationary point of the $\mathbf{IaaS}_{OPT}$ as $\mu \to 0$. To compute a solution we use Algorithm S presented in [10], (see Algorithm 1), which solves a sequence of problems $P(\mu)$. The algorithm stops when the Euclidean distance between two successive iterations is lower than a suitable threshold ϵ. We verified that in practice the algorithm converges very quickly. In the experiments described in the next section, the algorithm converged in no more than 3 iterations using $\epsilon = 10^{-4}$.

Algorithm 1 Algorithm S [10]

```
Let $\{\mu^k\}$ be any sequence of nonzero numbers with $\lim_{k\to\infty} \mu^k = 0$. Choose $w^0 = (\sigma^0, x^0, z^0, \lambda^0) \in \Re^{3N+l+l}$, and set $k = 1$
while $||e|| > \epsilon$ do
    Find a stationary point $w^k$ of $P(\mu^k)$
    $e = w^k - w^{k-1}$
    $k = k + 1$
end while
```

5 Experimental Results

In this section we investigate through numerical experiments the behavior of the proposed provisioning and pricing strategy. First, in Section 5.1, we compute the system equilibria in different scenarios, and study how flat, on demand and spot VMs are allocated among the competing SaaS providers and the associated spot prices under different workload and bidding configurations. Then in Section 5.2 we compare the proposed policy with the service provisioning and pricing policy studied in [4].

For the analysis, we implemented the algorithms presented in Section 4 as well as the algorithms presented in [4] in MATLAB. For the solution of the MPEC problem via Algorithm S, the parameter μ is initially set to 0.0001 and reduced by a factor of 100 at each iteration and the stopping parameter ϵ is set to 10^{-4}.

[4]Observe that for $\mu = 0$, if $s \in SOL(\sigma)$ is a solution of the original problem then either $g(s) < 0$ and $\lambda = 0$, in which case the constraint (22) reduces to $\sqrt{(z)^2} - z = 0$ or $g(s) = 0$, which implies that $z = 0$, and (22) reduces to $\sqrt{(-\lambda)^2} - \lambda = 0$.

5.1 Provisioning and Pricing Strategies Analysis

We consider one IaaS provider which sells his resources to ten SaaS providers. For the sake of simplicity we assume that each SaaS provider offers only one Web service. If not differently stated, we set $\mathcal{S} = 160$, $\varphi = 0.24\$$, $\delta = 1.24\$$, $f_u = 4$, $\mu_{u,1} = 10$ req/s, $m_{u,1} = -1$, $U_u^{max} = 0.9$, $\sigma^L = 0.15\$$ and $\sigma^U_{u,1} = 0.5\$$, for all $u \in \{1, \ldots, 10\}$. These parameters correspond to those adopted in [4] for the sake of comparison.

We first consider a homogeneous scenario with the SaaS providers parameters set as described above. In this scenario, by simmetry, all the SaaS providers obtain the same number of VMs and the same price for spot instances. The results are summarized in Table 2 for different values of the SaaS predicted load $\Lambda_{u,1}$ ranging from 20 req/s to 80 req/s.

$\Lambda_{u,1}$ (req/s)	20	40	60	80
flat	4	4	4	4
on demand	0	3.59	7.38	11.18
spot	2	3.59	4.61	0.81
spot price (\$)	0.25	0.31	0.36	0.5
available spot	120	80	46.1	8.1
unsold spot	100	44.1	0	0

Table 2: VMs allocation and spot price for increasing SaaS providers predicted load

In the first stage, the SaaS providers determine the amount of flat and on demand instances: in this example, flat instances are always used up to the maximum value $f_u = 4$, independently from the load $\Lambda_{u,1}$; on the other hand, the number of bought on demand instances grows from 0 to 11.18, because more instances are needed to satisfy the QoS constraints when the predicted load increases. This is reflected in the amount of unsold resources after the first stage, which decreases from 120 VMs when $\Lambda_{u,1} = 20$ down to 8.1 VMs when $\Lambda_{u,1} = 80$. In the second stage, the SaaS providers compete for the unsold IaaS capacity. When demand is high ($\Lambda_{u,1} = 80$), the IaaS provider is able to sell the small fraction of unused resources at the maximum price $\sigma^U_{u,1} = 5\$$. As demand decreases, the IaaS provider decreases the spot price, to sell more VMs and to increase his revenue. Observe that, when demand is low ($\Lambda_{u,1} \leq 40$) some capacity remains unsold even after the second stage since the SaaS providers reach equilibrium between the cost charged on the users (which is function of the response time) and the cost of additional VMs. At the same time, the IaaS provider has no incentive to further reduce the price. It is worth observing that, only when resources are scarce, the IaaS provider maximizes his revenue by charging the maximal price; otherwise, it is more profitable for the IaaS provider to lower the spot prices with respect to the maximal bid $\sigma^U_{u,k}$ and to sell additional VMs to users.

	$\Lambda_{u,1}$ (req/s)		$\sigma_{u,1}$ (\$)		$m_{u,1}$	
	20	80	0.5	0.3	-2	-1
flat	4	4	4	4	4	4
on demand	0	11.18	7.38	7.38	9.62	7.38
spot	2	7.18	3.65	5.57	4.09	2.89
spot price (\$)	0.25	0.31	0.43	0.3	0.5	0.5
spot available	64		46.1		34.9	
spot unsold	18.1		0		0	

Table 3: VMs allocation and spot prices for SaaS providers in heterogeneous scenarios

We now turn our attention to the provisioning and pricing solutions in heterogeneous scenarios. For the sake of simplicity, in the following examples, we will consider in each scenario only two different classes of users. We first consider the case where the SaaS providers have different predicted load. In the second column of Table 3 we show the results when half of the SaaS providers have a have a predicted load $\Lambda_{u,1} = 20$ req/s and the other half a predicted load $\Lambda_{u,1} = 80$ req/s. Observe that the number of purchased flat and on demand instances is the same as in the previous homogeneous example. This can be explained by observing in the first stage the SaaS providers independently determine the amount of flat and on demand instances to buy (under the implicit assumption that the IaaS provider has enough resources to allocate all the requested flat and on demand instances) the final allocation only depends on the provider parameters. Looking at the spot allocation and price, instead, we observe that while SaaS providers with the lower load are charged a relatively low price and buy only a small number of spot VMs, SaaS providers with the higher load are charged a higher price and buy more spot VMs. We observe that also in this case, at equilibrium, the IaaS provider maximizes his profit by charging less then the maximum price so that overall, the lower per VM profit is compensated by the higher volume of sold instances.

The third column of Table 3 shows the results when we consider two classes of SaaS providers with different maximum prices: five providers have $\sigma^U_{u,1} = 0.5\$$, while the rest of providers have $\sigma^U_{u,1} = 0.3\$$. We assume that the SaaS providers receive the same predicted load $\Lambda_{u,1} = 60$ req/s. Because all the providers have the same parameters, except for the bid which impacts only the second stage, they buy the same number of flat and on demand instances. However, as expected, given the different maximum price, the price and the number of purchased spot instances differ for the two classes of SaaS providers. It is interesting to compare these results to the first scenario we have considered, where all the SaaS providers had the same maximum price $\sigma^U_{u,1} = 0.5\$$; in that case, when the load was $\Lambda_{u,1} = 60$ req/s the optimal strategy for the IaaS provider was to set $\sigma_{u,1}$=0.36$ for all the users. In the current scenario, however, the maximum price for half of the SaaS providers is only 0.3$. So, it is no surprise that the optimal pricing strategy for the IaaS provider is to set $\sigma_{u,1}$=0.3$ for these providers, and a higher price, $\sigma_{u,1}$=0.43$, for the others. In other words, the revenue that is lost by the IaaS provider by selling spot instances at 0.3$ is recovered by increasing the spot price for those who bid 0.5$.

In the last scenario, we assume that all the SaaS providers have the same predicted load $\Lambda_{u,1} = 60$ req/s but different slopes of the utility function: specifically, half of the SaaS providers have a slope $m^U_{u,1}$ = -2, while the others have a slope $m^U_{u,1}$ = -1. This corresponds, for instance, to a scenario where half of the providers charge their users twice as much, all the rest being equal. The results are shown in the fourth column of Table 3. As in all the previous scenarios, all the SaaS providers buy 4 flat VMs. However, due to the different slopes, the providers which are characterized by a steeper utility, *i.e.*, $m_{u,1} = -2$, buy more on demand instances and spot instances. In this scenario, the IaaS provider can charge the maximum price 0.5$ to both as he is able to free all the unused resources at the maximum price.

5.2 Comparison with the Provisioning Scheme in [4]

We now compare the proposed provisioning and pricing scheme with the one presented in [4]. The authors of [4] study a provisioning problem very similar to the one presented in this paper. Differently from our two-stage allocation strategy, they consider a one stage provisioning problem, where, at the same time: the SaaS providers determine the number of flat, on demand and spot instances to buy as to maximize their revenue given the service SLA; and, the IaaS provider determines the spot instances price $\sigma_{u,k}$ as to maximize his profit, taking into account that each SaaS provider is characterized by a maximum cost $\sigma^U_{u,k}$ he is willing to pay for spot instance per hour. The conflicting situation is modeled as a GNEP and the service provisioning and pricing policy are derived from the game equilibrium. In particular, given the specific problem structure, they show that the dominant IaaS provider strategy consists in setting $\sigma_{u,k} = \sigma^U_{u,k}$, *i.e.*, in charging each SaaS user always the maximum price. They present a general solution method and evaluate the proposed scheme under different scenarios.

Despite the similarities, our provisioning scheme is substantially different: in our two stage approach, SaaS providers first buy only flat and on demand instances while spot instanced are only allocated in the second stage, with the IaaS provider determining the price as to maximize his profit. The resulting conflicting situation is modeled as a Stackelberg game where the IaaS provider takes the role of the leader. Comparing the two approaches, we expect that in our scheme the SaaS providers are more likely to buy a higher number of flat and especially of on demand instances, which are more expensive (but also more reliable as the IaaS provider cannot terminate them) since these type of instances are allocated first. This should result in higher cost for the SaaS provider and higher profit for the IaaS provider. Moreover, in the second stage, since competition for the spot instances is modeled as a Stackelberg game, we expect the IaaS provider to experience higher profits from the spot instances auction characterized, as observed in the previous examples, by lower than the maximum allowed on spot prices, larger volumes and higher overall profit.

SaaS	$\mu_{u,1}$	f^U_u	$\sigma^U_{u,1}$	SaaS	$\mu_{u,1}$	f^U_u	$\sigma^U_{u,1}$
1	11	5	0.38	6	12	3	0.23
2	5	5	0.49	7	12	3	0.44
3	13	3	0.16	8	8	4	0.3
4	14	5	0.83	9	11	5	0.54
5	11	4	0.28	10	6	5	0.42

Table 4: SaaS providers parameters

For the sake of comparison, we simulated a dynamic scenario using the two different policies. We considered 10 SaaS providers, each offering a single service. Every hour, each SaaS provider, given the forecasted load for the next hour determined the number and type of VMs to allocate while the IaaS provider determined the price of the spot instances. The predicted load $\Lambda_{u,1}$ of each SaaS provider is each time randomly generated uniformly in the interval $[20, 80]$ req/s. We set for all the providers, $R^{max}_{u,1} = 2s$, $C_{u,1} = 2\$$ (corresponding to $m_{u,1} = -1$) and $U^{max}_u = 0.9$. The other parameters are shown in Table 4 and were kept constant during the simulation. Because in [4] the variable s^U is fixed a priori and it is independent from the amount of flat and on demand instances sold, we fixed $s^U = 30$ for the whole simulation for both policies.

We run a simulation corresponding to a period of one week (168 hours).

Figure 1 shows the IaaS provider per hour revenue over time. As expected, the revenue obtained using our service provisioning and pricing policy is greater than the revenue obtained using the strategy presented in [4] (red curve, label GNEP).

	flat	on demand	spot	total
Stackelberg	6961.75	8706.31	5040	20708.06
GNEP	6961.75	3910.19	5040	15911.94
Stackelberg	1670.82$	10795.82$	1734.65$	14201.29$
GNEP	1670.82$	4848.63$	1575.48$	8094.93

Table 5: Total number of sold VMs and relative revenue

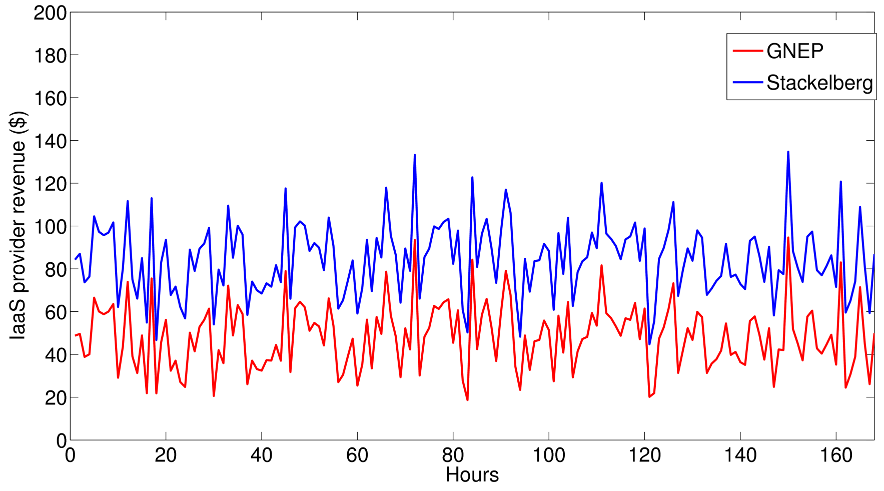

Figure 1: IaaS provider revenue using the two different strategies

Table 5 shows the breakdown of the number of allocated VMs and IaaS revenue per type of instance. As anticipated, our scheme results in a higher number of on demand instances (more than twice as much). The number of spot instances is the same but this is actually a consequence of having a fixed σ^{U}[5]. Nevertheless, our scheme yields higher spot VMs revenues to the IaaS provider. As shown above, this is a consequence of the fact that in our policy the IaaS provider, by setting a price that is lower than the the SaaS provider maximum bid, incentivates the SaaS providers to buy more spot instances, which results in overall higher IaaS profit.

SaaS	Stackelberg	GNEP	SaaS	Stackelberg	GNEP
1	73.96	75.55	6	75.44	78.5
2	56.1	55.76	7	79.96	80.31
3	78.04	80.79	8	73.4	78.75
4	81.79	81.54	9	74.86	74.49
5	71.36	73.84	10	62.2	64.86

Table 6: Average SaaS providers profit ($).

Tables 6-7 show, respectively, the profit, the cost and the average number of bought VMs for the different SaaS providers. We can observe that, indeed, our approach results in a higher number of VMs bought by the SaaS providers and a corresponding higher cost, which justifies the significant larger IaaS provider profits (+64%). Interestingly, the SaaS profits decrease only by a small fraction and in some cases (SaaS providers 2, 4 and 9) they even increase. This is not completely unexpected since as the number of VMs per service increases, the service response time decreases which in turn, given the SaaS revenue function, yields higher revenues.

[5]For a more detailed comparison we should have modified the model in [4] to reflect our scheme where the number of available spot instances depends on the number of allocated flat and on demand instances.

	Cost to buy VMs ($)		Number of bought VMs	
SaaS	Stackelberg	GNEP	Stackelberg	GNEP
1	5.60	2.53	9.99	7.87
2	18.15	15.15	21.20	17.09
3	5.82	1.35	10.76	6.96
4	4.02	3.47	7.58	6.63
5	6.08	1.98	10.37	7.66
6	6.47	1.71	10.47	7.31
7	7.16	5.28	9.5	7.52
8	11.28	3.30	15.76	11.54
9	5.71	4.83	9.43	7.90
10	14.20	8.53	18.15	14.18

Table 7: Average SaaS providers cost to buy VMs and number of bought VMs

6 Related Work

Game theory is a useful tool to deal with those situations where the interaction across players has to be taken into account. These situations cannot be handled with the classical optimization theory, because the action of each player affects not only the player itself, but the others players too. As a consequence, game theory can be successfully applied to typical ICT problems, like resource allocation, Quality of Service (QoS), pricing and load balancing. For example, it has been largely applied by the networking research community, as summarized in the survey [2], where the authors review different modeling and solution concepts of networking games as well as a number of different applications in telecommunications and wireless networks that can take advantage from game theory. Many studies, e.g., [6, 8], have proposed Game-theoretic methods to optimally solve the resource allocation problem in network systems from the viewpoint of resource owners. Load balancing in distributed systems has been also tackled through game theory, mostly exploiting solutions based on the Nash equilibrium concept, e.g. [14, 21].

Game theory has bee already applied to Cloud computing to deal mainly with resource allocation and pricing issues [4, 5, 15, 20, 23, 25]. A Bayesian Nash equilibrium allocation algorithm to solve the resource management problem has been proposed in [23]. A QoS-constrained resource allocation, where Cloud users submit intensive computation tasks has been proposed in [25]; however, only a single type of VM instances is considered.

The perspective of a SaaS provider is taken in [20], where the authors have provided a theoretical framework for resource management for Saas providers so they can efficiently control the service levels offered to their customers. They have formulated the problem as a non-cooperative game and proposed a resource bidding and allocation framework that can be viewed as a generalization of the Kelly mechanism [16].

The perspective of a IaaS provider is pursued in [15] by Hadji et al. to determine the optimal suggested prices by the Cloud provider and the optimal user demands. Their model consists in the cloud provider suggesting differentiated prices according to demand and users updating their requests in view of the proposed price. To this end, their game theoretical model is based on a Stackelberg game and the policy consists in finding the Stackelberg equilibrium. However, their model does not distinguish among flat, on demand, and spot instances and QoS constraints of SaaS providers are not taken into account as well.

As pointed out in Section 5.2 the works in [4, 5] by Ardagna et al. are most closely related to our proposal; we have already discussed how our proposal differs from their GNEP formulation and compared the two approaches, showing that our strategy results in a higher number of on demand instances and yields higher revenues for the IaaS provider.

Pricing and performance issues related to spot instances have been recently the subject of an increasing research interest, resulting in a consistent number of works that apply a variety of methodologies different from the game theoretical approach we deal with. For example, in [26] Yi et al. have studied how checkpointing and work migration strategies can be used to minimize the monetary cost and volatility of resource provisioning due to market dynamics. In [24] Voorsluys and Buyya have proposed a resource provisioning strategy that addresses the problem of running compute-intensive applications on intermittent spot instances tolerating their sudden unavailability, while also aiming to run applications in a fast and economical way. Those studies focus on helping end users to better use spot instances, which is different from our objective of maximizing the IaaS provider revenue while satisfying the QoS constraints of the SaaS providers.

Focusing on the IaaS provider perspective, in [28] Zhang et al. have faced the problem of determining an optimal resource allocation in such a way to optimize the IaaS revenue while minimizing energy cost through a constrained discrete-time finite-horizon optimal control formulation.

The problem of designing efficient bidding strategies has been considered in some works, e.g., [22, 27]. Zaman and Grosu have considered combinatorial auctions in the Cloud environment and their proposal guarantees the maximum utility for a user when he bids truthfully [27]. Tang et al. have proposed an optimal bidding strategy for spot instances in order to minimize the cost and volatility of resource provisioning and formulate their problem as a Constrained Markov Decision Process [22].

A reverse engineering analysis on how Amazon prices its spare capacity has been conducted in [1]; from the analysis it seems that Amazon sets the prices of EC2 spot instances at random from within a tight price interval via a dynamic hidden reserve price.

Although still only one IaaS provider offers spot VMs, many argue that market economies will be increasingly prevalent in order to achieve high utilization in data centers that are often under-utilized and the first public marketplaces for unused capacity are already on the scene. For example, SpotCloud [9] provides a spot market, where buyers can acquire from sellers excess or unused computing capacity with immediate delivery for their short term needs. Such a Cloud computing spot market, which is still in its infancy, opens up new research directions that can be worth of future investigation.

7 Conclusions

In this paper we presented a service provisioning and pricing strategy for a Cloud system based on a game theoretical approach. We considered several SaaS providers that offer a set of Web services with QoS constraints using the Cloud facilities provided by an IaaS provider. We proposed a two stage service provisioning policy: in the first stage, the SaaS providers buy VMs at a fixed price, while in the second stage they bid and compete to buy VMs instantiated on the unused IaaS provider capacity. The price of the spot instances is dynamically determined by the IaaS provider given the SaaS bids and with the aim of maximizing his revenue. While we solved the first stage by means of standard optimization techniques, we modeled the second stage conflicting situation as a Stackelberg game computed its equilibrium price and allocation strategy by solving an MPEC problem.

Our experimental results analyzed the proposed policy behavior under different scenarios. They revealed the ability of the IaaS provider to set a price lower than the bid to incentivize the SaaS providers to buy more instances. Furthermore, we compared our policy with the one presented in [4]. The numerical results showed that using our strategy the IaaS provider revenue increases, at the expense of a lower profit for the SaaS providers. However, the latter can offer more reliable services and better performance to their users.

In future research work we will address the development of a distributed version of the proposed service provisioning and pricing strategy, so that it can be profitably used in a real environment. Furthermore, we will investigate a different scenario in which the price fixed by the IaaS provider is the same for all the SaaS providers and will perform a comparison of these different strategies.

References

[1] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 spot instance pricing. In *Proc. of IEEE 3rd Int'l Conf. on Cloud Computing Technology and Science*, CloudCom'11, pages 304–311, Dec. 2011.

[2] 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.

[3] Amazon Web Services LLC. Amazon Elastic Compute Cloud (Amazon EC2), 2012. `http://aws.amazon.com/ec2/`.

[4] D. Ardagna, B. Panicucci, and M. Passacantando. A game theoretic formulation of the service provisioning problem in cloud systems. In *Proc. of 20th Int'l Conf. on World Wide Web*, WWW '11, pages 177–186, 2011.

[5] D. Ardagna, B. Panicucci, and M. Passacantando. Generalized Nash equilibria for the service provisioning problem in cloud systems. *IEEE Transactions on Services Computing*, 2013. To appear.

[6] J. Bredin, R. T. Maheswaran, c. Imer, T. Başar, D. Kotz, and D. Rus. A game-theoretic formulation of multi-agent resource allocation. In *Proc. of 4h Int'l Conf. on Autonomous Agents*, AGENTS '00, pages 349–356, 2000.

[7] J. Cardinal, M. Labbe, S. Langerman, and B. Palop. Pricing geometric transportation networks. *Int'l J. of Computational Geometry & Applications*, 19(6):507–520, 2009.

[8] S. Dobzinski, N. Nisan, and M. Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. In *Proc. of 37th Annual ACM Symposium on Theory of Computing*, STOC '05, pages 610–618, 2005.

[9] Enomaly Inc. SpotCloud - Cloud capacity clearing house, 2012. `http://spotcloud.com/`.

[10] F. Facchinei, H. Jiang, and L. Qi. A smoothing method for mathematical programs with equilibrium constraints. *Mathematical Programming*, 85:107–134, 1999.

[11] F. Facchinei and C. Kanzow. Generalized Nash equilibrium problems. *4OR: A Quarterly Journal of Operations Research*, 5:173–210, 2007. 10.1007/s10288-007-0054-4.

[12] F. Facchinei and J.-S. Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems, Vol. I.* Springer, New York, 2003.

[13] D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, Cambridge, MA, 1991.

[14] D. Grosu and A. T. Chronopoulos. Noncooperative load balancing in distributed systems. *Journal of Parallel and Distributed Computing*, 65(9):1022 – 1034, 2005.

[15] M. Hadji, W. Louati, and D. Zeghlache. Constrained pricing for cloud resource allocation. In *Proc. of 10th IEEE Int'l Symp. on Network Computing and Applications*, NCA '11, pages 359–365, Aug. 2011.

[16] F. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8(1):33–37, 1997.

[17] Y. A. Korilis, A. A. Lazar, and A. Orda. Achieving network optima using stackelberg routing strategies. *IEEE/ACM Trans. Netw.*, 5(1):161–173, Feb. 1997.

[18] H. Liu. Cutting MapReduce cost with spot market. In *Proc. of 3rd USENIX Conf. on Hot topics in Cloud Computing*, HotCloud'11, 2011.

[19] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming.* Springer, New York, 2008.

[20] R. T. Ma, D. M. Chiu, J. C. Lui, V. Misra, and D. Rubenstein. On resource management for Cloud users: a generalized Kelly mechanism approach. Technical report, Dept. Computer Science, Columbia University, May 2010.

[21] S. Suri, C. D. Toth, and Y. Zhou. Selfish load balancing and atomic congestion games. *Algorithmica*, 47:79–96, 2007.

[22] S. Tang, J. Yuan, and X.-Y. Li. Towards optimal bidding strategy for Amazon EC2 cloud spot instance. In *Proc. of IEEE 5th Int'l Conf. on Cloud Computing*, pages 91–98, 2012.

[23] F. Teng and F. Magoulès. A new game theoretical resource allocation algorithm for cloud computing. In P. Bellavista, R.-S. Chang, H.-C. Chao, S.-F. Lin, and P. Sloot, editors, *Advances in Grid and Pervasive Computing*, volume 6104 of *Lecture Notes in Computer Science*, pages 321–330. Springer, 2010.

[24] W. Voorsluys and R. Buyya. Reliable provisioning of spot instances for compute-intensive applications. In *Proc. of IEEE 26th Int'l Conf. on Advanced Information Networking and Applications*, AINA'12, pages 542–549, Mar. 2012.

[25] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *J. Supercomput.*, 54(2):252–269, Nov. 2010.

[26] S. Yi, A. Andrzejak, and D. Kondo. Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *IEEE Transactions on Services Computing*, 5(4):512–524, 2012.

[27] S. Zaman and D. Grosu. Efficient bidding for virtual machine instances in clouds. In *Proc. of 2011 IEEE Int'l Conf. on Cloud Computing*, CLOUD'11, pages 41–48, july 2011.

[28] Q. Zhang, E. Gürses, R. Boutaba, and J. Xiao. Dynamic resource allocation for spot markets in clouds. In *Proc. of 11th USENIX Conf. on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'11, 2011.