Bidding Strategies in QoS-Aware Cloud Systems Based on N-Armed Bandit Problems

Marco Abundo*, Valerio Di Valerio†, Valeria Cardellini‡ and Francesco Lo Presti§
Department of Civil Engineering and Computer Science Engineering
University of Roma Tor Vergata, Italy
marco.abundo@gmail.com, †di.valerio@ing.uniroma2.it, ‡cardellini@ing.uniroma2.it, §lopresti@info.uniroma2.it

***Abstract*—In this paper we consider a set of 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 put our focus on the bidding decision process by the SaaS providers, which takes place during the second stage, and apply N-armed bandit problems, in which the player is faced repeatedly with a choice among N different options, and every time he submits his decision evaluating past feedbacks. Through numerical experiments, we analyze proposed strategies under different scenarios and prove the SaaS providers ability to refine their behavior round by round and to determine the best bid so to maximize their revenue and achieve as many spot resources as possible, also addressing the importance of a trade-off between exploration and exploitation, i.e., among greedy and non-greedy actions.**

***Keywords*-Cloud computing, N-armed bandit problems, provisioning, QoS, SLA, auctions**

I. Introduction

In the past few years, cloud computing has been experiencing an ever increasing growth. More and more companies are taking advantage of commercially available services provided by Infrastructure as a Service (IaaS) providers, that allow to scale up or down computing resources as demand changes by allocating or deallocating rapidly virtual computing and storage resources. IaaS services let Software as a Service (SaaS) providers offer fresh applications, with no need to maintain underlying infrastructures.

IaaS providers usually make resources available as Virtual Machine (VM) instances to customers, and this process is often governed by a pay-as-you-go model on a per-hour basis at a fixed price (*on demand* VMs).

If resource utilization can be forecasted in advance, then IaaS customers can also reserve *flat* VMs, paying a long-term reservation fee in addition to a per-hour price depending on the effective resource usage, of course cheaper than the on demand price. Since flat and on demand instances do not saturate the IaaS provider capacity, he can sell the spare capacity as *spot* VMs via an auction.

For SaaS providers, spot instances provide cost-effective means to deal with unexpected load spikes and support high computation-demanding applications, but at the risk of a lower reliability than flat and on demand instances, since the IaaS provider can deallocate spot instances without notice due to price and demand fluctuations. For example, Amazon's Elastic Cloud Computing (EC2) service offers three distinct types of VM instances (i.e., flat, on demand, and spot VMs) [1], governed by different fees and reliability.

In this paper we consider a Cloud environment in which an IaaS provider offers flat, on demand, and spot VM instances to a set of SaaS providers. In turn, SaaS providers make available to their end users Web services with Quality of Service (QoS) constraints, through the IaaS resources used to run the offered applications. Revenues and penalties of each SaaS provider are regulated according to the respect of an agreed performance level, specified in a Service Level Agreement (SLA) contract stipulated with end users. As a consequence, SaaS providers have to address the issue of allocating the optimal number of VMs to satisfy their SLAs while still maximizing their revenues.

We consider a two-stage resource provisioning and pricing strategy. In the first stage, each SaaS provider independently requests the number of flat and on demand VMs determined by solving an optimization problem. In the second stage, all the SaaS providers take part into an auction for the still available IaaS provider capacity offered as spot VMs. The bidding mechanism requires each SaaS provider to specify the maximum price he is willing to pay for a spot VM, and the number of requested VMs; then, the IaaS provider computes the spot price. When the SaaS provider bid is lower than the spot price, he will get no resource. Otherwise, the SaaS provider will receive all requested resources, paying them at the price set by the IaaS provider.

We study the SaaS provider bidding strategy under the realistic assumption that, at each step, a SaaS user has no knowledge of the behaviour of the other users, their number, as well as the amount of spare resources the IaaS provider wants to sell. Hence, we assume each SaaS provider can only *learn* the bidding strategy through experience over

time. To this end, we resort to reinforcement learning (RL) techniques [2].

Specifically, we formulate the bidding strategy as an N-armed bandit problem [2]. In N-armed bandit problems, the player is repeatedly faced with a choice among N different actions, which in our setting represent possible bids, and, every time he submits his decision, he receives a numerical reward (or cost/regret) depending on the performed decision. Such type of problems is particularly suitable to our setting, since it requires no knowledge of the environment and its dynamics. By observing the incurred regret depending on the chosen action, the SaaS provider learns over time the bidding strategy which minimizes the cost by judiciously trading off exploitation of the past experience (the greedy choice of the immediately best option) and exploration (selection of non-greedy, hence possibly suboptimal actions). To the best of our knowledge, N-armed bandit problems have not yet been applied to cloud resource allocation problems.

We study the effectiveness of the resulting bidding strategies through simulation experiments under different settings. Specifically, we study the SaaS provider behaviour under different IaaS provider pricing policies, namely clearing market, constant, random or a combination of the above approaches. We also consider the special case in which the IaaS provider himself uses a N-armed bandit approach to determine his pricing scheme. We also explore the trade-off between exploration and exploitation.

Our results reveal that, through the appliance of N-armed bandit problems and a thorough trade-off between exploration and exploitation, SaaS providers succeed, round by round, in refining their behaviour and allocating as many spot resources as possible by means of the best bid.

The rest of the paper is organized as follows. In Section II we review some related research efforts, while in Section III we introduce N-armed bandit problems. In Section IV we provide the problem statement and define the system model. We describe the solution to the bidding problem, based on the spot instances sale through an auction, in Section V. In Section VI we analyze through numerical experiments the behavior of the proposed strategies. Finally, we draw some conclusions and give hints for future work in Section VII.

II. Related Work

A considerable number of research efforts have focused on resource allocation policies in dynamic environments and on the adoption of auction mechanisms. Mastronarde and van der Schaar addressed in [3] the issue of providing media services to multiple autonomous wireless users at the edge of a content delivery network in a setting where resources are priced on the basis of real-time market demands. They adopted the progressive second price auction mechanism and dealt both with learning solutions that require an agent to collect information about his opponents' bids, and a greedy solution that does not rely on it, as in this paper. Another common point with our work is that no strict state model is applied, and this is particularly important for actual dynamic scenarios in which few hypotheses can be taken.

Two recent proposals in [4], [5] adopt action-based mechanisms to deal with spot instances. In [4] Song et al. proposed a mechanism based on a repeated uniform price auction and proved its truthfulness. To achieve a better quality of service, they exploited the flexibility of adjusting bids during job execution and discussed a related bidding adjustment model. They showed that a uniform price action achieves optimal efficiency among all single-price auctions in a data center spot market. In [5] Wang et al. proposed dynamic auctions where computing instances are periodically auctioned off to accommodate user demands over time. They addressed the two main challenges of revenue maximization and auction truthfulness, and encompassed an allocation scheme to determine the amount of instances to be auctioned off in each period, based on the approximated solution of an analytical MDP, differently from our paper.

Learning strategies alternative to our approach were also proposed in [6] (defining competitive behavior as a conjectural equilibrium), [7] (focusing on cooperation strategies), [8] (based on a stochastic game framework) and [9], [10] (optimizing performance of distributed platforms via bandit-based RL algorithms).

An interesting reverse engineering analysis of the price of Amazon EC2 spot instances was conducted by Ben-Yehuda et al. [11]. By analyzing the spot price history, they built a model generating prices consistent with existing price traces. Their findings suggest prices are not market-driven (differently from what stated in [12], [13]), but are rather likely generated most of the time at random within a tight price range via a dynamic hidden reserve price mechanism.

N-armed bandit problems [14], [15] have been applied to many settings, but to the best of our knowledge they have been never applied to cloud resource allocation scenarios, as done in this paper. In [16], Kalathil et al. studied the problem of distributed online learning with multiple players in multi-armed bandits and proposed an online index-based distributed learning policy. In [17], N-armed bandits have been applied in pay-per-click auctions for Internet advertising, while in [18] for truthful sponsored search auctions and in [19] for keywords selection by search-based advertising.

III. Preliminaries: N-Armed Bandit Problems

The N-armed bandit problem is the problem a decision maker faces when deciding, *repeatedly*, which action to play among a set of N actions. Each action has a mean cost associated to it and the player goal is to minimize the sum of the costs paid through a sequence of actions. However, the player does not know the cost associated to each action, otherwise the problem would be trivial, but it is assumed he can estimate such value. Through subsequent rounds, the

player aims at refining his estimation to learn which are the best actions and concentrate his choice on them.

Suppose that at time t action a_i is chosen for the n-th time and the resultant cost is $q_n(a_i)$. The estimation of the mean value of a_i can be simply defined as follows:

$$Q_t(a_i) = \frac{\sum_{j=1}^{n} q_j(a_i)}{n} \quad (1)$$

At any time there is at least one action whose estimated value $Q_t(a_i)$ is the lowest: the greedy action. When the player chooses it, then he *exploits* his experience. If instead the player selects one of the non-greedy actions, then we say he is *exploring*: he is improving the estimation $Q_t(a_i)$ of the non-greedy action's value. Exploitation is the best way to minimize the expected cost on a single round, while exploration lets refine the experience information base and achieve a lower cost on the long run. It is clear how to learn the best actions and still minimize paid costs requires to carefully trade-off between exploitation and exploration.

A common approach to get at this is to use the ϵ-greedy heuristic [2], switching many times from selecting the greedy action and taking one of the others. Formally, the player acts greedily with probability $1 - \epsilon$ and non-greedily with probability ϵ, choosing a random non-greedy action from a uniform distribution. This lets, on the long run, update estimated action costs and re-think wrong approaches that seemed effective. We adopted, among others, an ϵ-greedy strategy because, as stated in [2], despite its simplicity, it is still the best among practically useful algorithms.

N-armed bandit problems are a basic but effective variant of *reinforcement learning* [2], particularly suitable to settings in which the action performed in the current decision round has no influence on future costs, system states, and decisions. Indeed, evaluative feedback in N-armed bandit problems only looks at the past but it is not forward-looking like in reinforcement learning, i.e., it does not take into account future rounds and future states, but only the current situation.

IV. System Model

We consider a set U of SaaS providers, each one offering a set of Web services/applications A_u, $u \in U$ by taking advantage of the VM resources offered by an IaaS provider. Let each Web service $k \in A_u$ be featured by a *SLA* that determines its QoS levels, i.e., the service response time, and the associated cost/penalty for its usage. The Web services are hosted on virtual machines instantiated by the IaaS provider. For the sake of simplicity, we assume the IaaS provider offers only one size of VMs, so that all VMs have the same capacity. We further assume that each VM hosts only one Web service. In turn, the same service can be hosted on multiple VMs; in that case, the workload is evenly shared among hosting VMs. The single-layer modeling, although with horizontal scaling, is a simplification, because modern applications are based on front-end and back-end logics, with different specialized layers.

The IaaS provider can offer to his users up to S VMs, selling them as flat, on demand, and spot instances. Flat instances are featured by a one-time payment plus a payment of φ units per hour of actual use. On demand instances have no one-time payment and are sold at a price δ, strictly greater than φ. Finally, spot instances are sold at a price σ. However, their price σ is not fixed, rather it varies over time according to users bids and the IaaS provider pricing strategy.

Each SaaS provider determines for each Web 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 application SLA. We assume the SaaS providers' SLA consists of an upper bound on the application response time $R_{u,k}$. Moreover, we assume the user per-request cost $\mathcal{C}_{u,k} = C_{u,k}\left(1 - \frac{R_{u,k}}{R_{u,k}^{max}}\right)$ to be a decreasing linear function of $R_{u,k}$, that becomes negative (hence, the SaaS provider incurs a penalty) when $R_{u,k} > R_{u,k}^{max}$. We denote with $m_{u,k} = -\frac{C_{u,k}}{R_{u,k}^{max}}$ the slope of this function. Modeling the cost as a linear function of the response time is an approximation: a fixed penalty whenever the maximum response time is exceeded would be more realistic; in addition, usually revenue does not scale linearly since stocks may give rise to discounts. Nevertheless, this allows us: (1) to consider a soft constraint on the response time, which enables the SaaS provider to trade-off revenues and infrastructural costs [20], [21]; (2) to model the SaaS providers gain to let them buy more VMs than the bare essential in order to satisfy the SLA, to increase the application reliability and performance and, in turn, their reputation.

We model each Web service 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 application, we have the following general expression for the per-hour SaaS profit (see [20], [21]); the first term represents the SaaS provider revenue, while the others the cost to use the IaaS provider facilities:

$$\Theta_u = \sum_{k \in A_u} \frac{m_{u,k}\Lambda_{u,k}\left(f_{u,k} + d_{u,k} + s_{u,k}\right)}{\mu_{u,k}\left(f_{u,k} + d_{u,k} + s_{u,k}\right) - \Lambda_{u,k}} + \\ -\varphi \sum_{k \in A_u} f_{u,k} - \delta \sum_{k \in A_u} d_{u,k} - \sigma \sum_{k \in A_u} s_{u,k} \quad (2)$$

A. Provisioning Scheme

We assume that every hour SaaS providers allocate and deallocate VMs relying on a future workload prediction. In this paper we follow the approach proposed in [21] and we consider a two-stage allocation procedure. In the first one, each SaaS provider independently determines, for each offered service, the number of flat and on demand instances[1]

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

which guarantees the performance level defined in the SLA to his prospective users and maximizes his profit. The second stage is devoted to the spot instances sale. We assume the IaaS provider implements some auction mechanism. Hence, SaaS providers compete for additional resources by submitting to the IaaS provider the bid, which defines the maximum per VM price they are willing to pay, and the number of requested VMs. In the next section we present the proposed bidding strategy, while in the remaining of this section we describe how a SaaS provider can compute, given the chosen bid, the number of VMs to be acquired.

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 that are needed to sustain the predicted load for the next hour while maximizing his profit. We assume that the IaaS provider has always enough resources to accomodate the SaaS providers demand. For each SaaS provider $u \in U$ we have the following optimization problem, as explained in [21]:

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

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

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

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

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 that their utilization is less than a threshold U_u^{max}. Note that, as in [20], [21], we deal with a relaxation of the real problem, because we do not impose to the variables to be integers. Otherwise, the problem would be too difficult to solve (i.e., NP-hard).

2) Second Stage: Spot Instances Sale: The second stage is devoted to the spot instances sale. The SaaS providers can increase their revenues by acquiring additional resources, so that the service response time may decrease paying a price lower than that for on demand VMs. Spot instances are made available by the IaaS provider via a bidding mechanism. The latter requires each SaaS provider to specify the maximum price σ_u he is willing to pay for a spot VM, and the number of requested VMs $s_u^* = \sum_{k \in \mathcal{A}_u} s_{u,k}^*$. Assuming a given bid σ_u, $s_{u,k}^*$ can be computed by maximizing $\bar{\Theta}_u$, which is obtained from Θ_u by replacing variables $f_{u,k}$ and $d_{u,k}$ with the number of flat ($\bar{f}_{u,k}$) and on demand ($\bar{d}_{u,k}$) VMs purchased in the first stage:

$$\bar{\Theta}_u = \sum_{k \in A_u} \frac{m_{u,k}\Lambda_{u,k}(\bar{f}_{u,k} + \bar{d}_{u,k} + s_{u,k})}{\mu_{u,k}(\bar{f}_{u,k} + \bar{d}_{u,k} + s_{u,k}) - \Lambda_{u,k}} - s_{u,k}\sigma_u \tag{7}$$

V. Bidding Approach

We model the bidding problem as an N-armed bandit problem. We describe in Section V-A how we model the key elements for N-armed bandit problems: the actions set and the cost associated to each action. Then, in Section V-B we discuss how to deal with a non stationary environment, when average costs associated to each action varies over time.

A. N-armed Bandit Model

1) Actions: Let A be a discrete set of N actions, such that $A = \{a_1, a_2, ..., a_N\}$. Each action a_i is a couple, i.e., $a_i = (\sigma_u, s_u^*)$, where σ_u is the bid and s_u^* the number of requested spot VMs, computed by solving (7).

2) Costs: We model the cost associated to a chosen action as the regret with respect to a desired optimal situation. We remark that this cost is unknown a priori but depends, time by time, on the auction outcome, i.e., the spot price $\hat{\sigma}$.

Given the spot price, the regret can be evaluated by leveraging the player desire to optimize the number of allocated spot VMs. When $\sigma_u < \hat{\sigma}$, then the player obtains no resource, and the regret is full. On the contrary, if $\sigma_u \geq \hat{\sigma}$, then the player will receive all the s_u^* resources paying them $s_u^*\hat{\sigma}$, and there are two possible situations:

- the player has no regret, since $\sigma_u = \hat{\sigma}$ and he requested exactly the number of resources he could obtain;
- the player has some regret: the estimation of the spot price (his bid) was greater than the real $\hat{\sigma}$. Therefore, despite he is paying $\hat{\sigma}$, he has demanded a number of resources lower that that he could have required.

The optimal number of resources s_u^o that the player u could have requested can be computed by maximizing $\bar{\Theta}_u$ with a bid of $\hat{\sigma}$ as input. Hence, the single cost ψ_u, at every round, is given by the percentage of resources that could have been obtained but were not:

$$\psi_u(a_i) = \frac{s_u^o - s_u^*}{s_u^o}, 0 \leq \psi_u \leq 1 \tag{8}$$

Let $\eta_u^{a_i}$ be the number of times player u has so far chosen action a_i and $\psi_{u_j}(a_i)$ the cost experienced by player u the j-th time he selected option a_i. The estimated cost for a_i is given at every round t by the average observed cost:

$$\Psi_{u_t}(a_i) = \frac{\sum_{j=1}^{\eta_u^{a_i}} \psi_{u_j}(a_i)}{\eta_u^{a_i}} \tag{9}$$

B. Discounted Value of Experience

The N-armed bandit method discussed above suits well in stationary environments, but becomes unsuitable if the bandit changes its working principles over time. When the IaaS provider always applies the same price selection policy, e.g., a constant spot price strategy or a stable market driven strategy, then the hypothesis of a stationary environment can be held, as shown by the experimental results in Section VI. On the other hand, when the feedback from the external environment is quite heterogeneous, the reinforcement learning problem cannot be approximated to an effectively stationary one, and, as stated in [2], the player needs to weight recent

regrets more than long-past ones. This can be achieved via a discount parameter α, with $0 \leq \alpha \leq 1$. In this case,

$$\Psi_{u_{t+1}}(a_i) = \alpha \Psi_{u_t}(a_i) + \psi_{u_{t+1}}(a_i) \quad (10)$$

where $\psi_{u_{t+1}}(a_i)$ is equal to 0 both when there is no regret and when action a_i has not been performed. Hence, $\Psi_{u_{t+1}}(a_i)$ is a weighted average of past regrets:

$$\Psi_{u_{t+1}}(a_i) = \alpha \Psi_{u_t}(a_i) + \psi_{u_{t+1}}(a_i) = ... = \sum_{j=0}^{t+1} \alpha^{t-j+1} \psi_{u_j}(a_i) \quad (11)$$

VI. Experimental Results

In this section we investigate through numerical experiments the behavior of the proposed bidding strategy. In order to analyze our strategy against several IaaS provider pricing policies, we examine different scenarios. We consider market-driven, constant, random, and strategic pricing policies, and a turnover of all of them. We measure the effectiveness of the bidding strategy through the average ratio between the number of spot resources achieved by SaaS providers (s_u^*) and the optimal value that could be obtained (s_u^o); the higher this value, belonging to [0, 1], the smaller the mean regret. Furthermore, we also consider the fraction of times that each SaaS provider selects a particular bid and the IaaS provider applies a given spot price.

We consider 10 SaaS providers, each of whom applies the proposed bidding strategy, and assume that each SaaS provider offers only one Web service (i.e., $k = 1$) and the IaaS provider has potentially infinite resources to allocate. Since we focus on the bidding process for spot instances, all the considered scenarios have the same parameters for flat and on demand VMs allocation, inspired to the costs of real IaaS providers and corresponding to those already adopted in [20], [21]. Specifically, the following parameters hold in all the settings: $\varphi = 0.24\$$, $f_u = 4$, $\delta = 1.24\$$, $\mu_{u,1} = 0.9$ req/s, $m_{u,1} = -1$, $U_u^{max} = 0.9$, $\Lambda_{u,k}$ uniformly distributed in [30, 80] req/s and changing at every round. For all scenarios, 30 different runs were performed. The experience was reset at every run, while, on the other hand, statistics about actions and resources were kept. In all cases, SaaS providers can select among 5 different options: 0.1\$, 0.2\$, 0.3\$, 0.4\$, 0.5\$.

A. Community and Market-driven Approach

In the first scenario the IaaS provider takes the decision about the threshold $\hat{\sigma}$ (the spot price paid by the Saas providers for a resource if they submitted a bid greater or equal) according to a *market-driven* strategy. In other words, he chooses $\hat{\sigma}$ s.t. he maximizes his per-hour revenue:

$$\hat{\sigma} = argmax_\sigma \sum_{u \in U} \sigma s_u^* \quad (12)$$

According to the ϵ-greedy approach, when exploring, players take a random action with probability 0.05, selecting an option based on a uniform distribution. ϵ is set to 0.05 for all SaaS providers. Table I reports how many times in percentage the SaaS providers selected each option, Table II depicts the IaaS provider behavior and how frequently he chose the five options as $\hat{\sigma}$, and Fig. 1 illustrates the effectiveness of the bidding strategy.

Table I
SCENARIO A: SAAS PROVIDER BEHAVIOR

	σ				
SaaS Provider	0.1\$	0.2\$	0.3\$	0.4\$	0.5\$
1	0.9261	0.0435	0.0101	0.0102	0.0101
2	0.9262	0.0436	0.0101	0.0100	0.0101
3	0.9279	0.0419	0.0101	0.0101	0.0100
4	0.9280	0.0419	0.0101	0.0101	0.0099
5	0.9278	0.0419	0.0100	0.0101	0.0103
6	0.9278	0.0420	0.0102	0.0101	0.0100
7	0.9278	0.0420	0.0102	0.0100	0.0100
8	0.9281	0.0418	0.0101	0.0100	0.0100
9	0.9252	0.0445	0.0101	0.0102	0.0101
10	0.9277	0.0416	0.0105	0.0101	0.0102

Table II
SCENARIO A: IAAS PROVIDER BEHAVIOR

0.1\$	0.2\$	0.3\$	0.4\$	0.5\$
0.966566	0.03337	0.000025	0.000025	0.000014

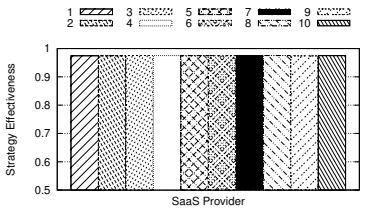

Figure 1. Scenario A: Strategy effectiveness (average ratio $\frac{s_u^*}{s_u^o}$)

As Fig. 1 and Tables I and II show, all players achieved more than 97.4% of the optimal amount of spot resources, still selecting, on average, over 92.5% of times the minimum bid. As a result, in over 96.6% of times, $\hat{\sigma}$ is equal to 0.1\$. Such an interesting result is only at a first glance surprising and stems from a sort of emergent behavior of the community, homogeneous and willing to minimize its regret. This situation motivates quite easily why a market-driven strategy could turn to be unprofitable for an IaaS provider and why, indeed, Amazon EC2 does not apply it [11].

B. Constant Spot Price

In the second scenario, the IaaS provider applies the same spot price, $\hat{\sigma} = 0.3$, independently of the SaaS providers bids. As in Scenario A, all SaaS providers adopt the ϵ-greedy approach, with ϵ set to 0.05. Table III reports how many times in percentage SaaS providers selected each option.

As expected, the learning approach lets the SaaS providers find that the best approach simply lies in bidding 0.3\$. As Fig. 2 shows, players bid the optimal value $\approx 96\%$ of times

Table III
SCENARIO B: SAAS PROVIDER BEHAVIOR

	σ				
SaaS Provider	0.1$	0.2$	0.3$	0.4$	0.5$
1	0.0100	0.0100	0.9599	0.0101	0.0100
2	0.0101	0.0099	0.9598	0.0101	0.0100
3	0.0101	0.0100	0.9596	0.0102	0.0101
4	0.0102	0.0102	0.9593	0.0102	0.0101
5	0.0099	0.0099	0.9601	0.0100	0.0100
6	0.0100	0.0099	0.9598	0.0101	0.0102
7	0.0100	0.0100	0.9600	0.0101	0.0099
8	0.0100	0.0100	0.9597	0.0102	0.0101
9	0.0099	0.0101	0.9602	0.0100	0.0098
10	0.0100	0.0099	0.9600	0.0102	0.0100

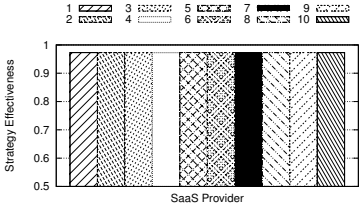

Figure 2. Scenario B: Strategy effectiveness (average ratio $\frac{s^*_u}{s^o_u}$)

(because of the ϵ-greedy approach, 5% of the times a random bid is selected, which on occasion results into the right bid).

C. Random Spot Price

In the third scenario, the IaaS provider follows a *random spot price* strategy, selecting a random option, independently on the choice of SaaS providers. As in Scenarios A and B, each player applies the ϵ-greedy strategy where $\epsilon = 0.05$. Table IV reports how many times in percentage SaaS providers selected each option, while the IaaS provider behavior is obviously purely random, with a uniform distribution.

Table IV
SCENARIO C: SAAS PROVIDER BEHAVIOR

	σ				
SaaS Provider	0.1$	0.2$	0.3$	0.4$	0.5$
1	0.0102	0.0101	0.0106	0.0238	0.9453
2	0.0101	0.0102	0.0103	0.0172	0.9523
3	0.0101	0.0102	0.0106	0.0326	0.9365
4	0.0101	0.0101	0.0102	0.0170	0.9525
5	0.0101	0.0101	0.0112	0.0156	0.9531
6	0.0101	0.0101	0.0107	0.0259	0.9432
7	0.0101	0.0102	0.0105	0.0200	0.9493
8	0.0101	0.0102	0.0102	0.0174	0.9522
9	0.0101	0.0101	0.0105	0.0213	0.9480
10	0.0099	0.0102	0.0103	0.0150	0.9547

As Fig. 3 and Table IV show, the players receive 57.6% of the optimum, which is expected since the price is random. The players learn that the best bid is the maximum value, 0.5$ bid: there is no advantage to bid less than the maximum since no pricing criterion can be identified (being it random).

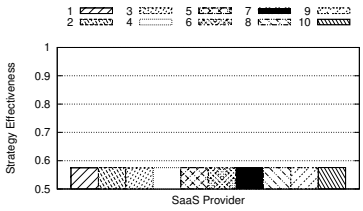

Figure 3. Scenario C: Strategy effectiveness (average ratio $\frac{s^*_u}{s^o_u}$)

D. N-armed Bandit Provisioning Strategy

In this scenario, differently from the previous ones, the IaaS provider also sets the price using an N-armed bandit ϵ-greedy approach, basically choosing the price that maximizes his revenue (given prior experience). This way, over time, the IaaS and SaaS providers influence each others.

Table V reports how many times in percentage the SaaS providers selected each option. Both the IaaS and SaaS providers are characterized by $\epsilon = 0.05$.

Table V
SCENARIO D: SAAS PROVIDER BEHAVIOR

	σ				
SaaS Provider	0.1$	0.2$	0.3$	0.4$	0.5$
1	0.0414	0.3576	0.0739	0.4517	0.0753
2	0.0414	0.3575	0.0739	0.4487	0.0785
3	0.0414	0.3579	0.0738	0.4527	0.0742
4	0.0418	0.3576	0.0729	0.4511	0.0766
5	0.0416	0.3578	0.0729	0.4498	0.0779
6	0.0417	0.3559	0.0749	0.4526	0.0749
7	0.0417	0.3550	0.0759	0.4512	0.0761
8	0.0417	0.3572	0.0743	0.4514	0.0754
9	0.0416	0.3573	0.0741	0.4519	0.0752
10	0.0417	0.3574	0.0739	0.4508	0.0761

Table VI presents the IaaS provider behavior, that principally focused on thresholds 0.2$ and 0.4$ as the most profitable according to past experience. The distribution of the IaaS provider actions is interestingly similar to that achieved by the SaaS providers.

Table VI
SCENARIO E: IAAS PROVIDER BEHAVIOR

0.1$	0.2$	0.3$	0.4$	0.5$
0.0418	0.3584	0.0734	0.4531	0.0733

As Fig. 4 shows, all players managed at obtaining more than 94% of the optimum. Nevertheless, thanks to the reciprocal influence among the IaaS provider and the SaaS providers, the average spot price is 0.3534$ and is not kept on the lower bound as in Scenario A. In fact, the partial blindness by the IaaS provider on current bids avoids an emergent convergence by the community to the 0.1$ bid.

E. High Varying Provisioning Strategy

Differently from previous scenarios, now the IaaS provider does not always apply the same price selection

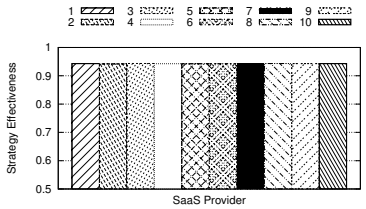

Figure 4. Scenario D: Strategy effectiveness (average ratio $\frac{s^*_u}{s^o_u}$)

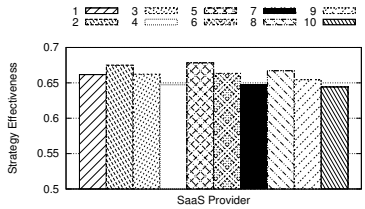

Figure 5. Scenario E: Strategy effectiveness (average ratio $\frac{s^*_u}{s^o_u}$)

Table VII
SCENARIO E: VALUES OF ϵ AND α

	SaaS Provider									
	1	2	3	4	5	6	7	8	9	10
ϵ	0	0	0.05	0.1	0	0.05	0.1	0	0.05	0
α	1	0.9999	0.9999	0.9999	0.999	0.999	0.999	0.99	0.99	0.9

policy, but he alternates every 300 hours between the market driven, the constant spot price, and the random spot price approaches. In order to face instability, players also consider a discount exponent α while carrying on the ϵ-greedy policy. Table VII reports values of ϵ and α for all players.

Table VIII reports how many times in percentage SaaS providers selected each option, while Table IX presents the IaaS provider behavior.

Table VIII
SCENARIO E: SAAS PROVIDER BEHAVIOR

	σ				
SaaS Provider	0.1$	0.2$	0.3$	0.4$	0.5$
1	0.0002	0.1586	0.0004	0.0676	0.7732
2	0.0004	0.1636	0.0035	0.1709	0.6616
3	0.0102	0.1705	0.0115	0.2120	0.5958
4	0.0204	0.1719	0.0202	0.2484	0.5391
5	0.0015	0.1790	0.0049	0.2794	0.5353
6	0.0100	0.1818	0.0108	0.2822	0.5152
7	0.0200	0.1835	0.0207	0.2960	0.4797
8	0.0080	0.1935	0.0151	0.3379	0.4455
9	0.0153	0.1917	0.0233	0.3298	0.4399
10	0.0478	0.1893	0.0669	0.3173	0.3787

Table IX
SCENARIO E: IAAS PROVIDER BEHAVIOR

0.1$	0.2$	0.3$	0.4$	0.5$
0.1336	0.1966	0.1334	0.2581	0.2783

Fig. 5 shows that a discount exponent close to 1 yet less than 1 lets achieve a better performance. On the other hand, in presence of discount, an ϵ less than 1 seems not to be profitable. This is likely due to the minor and minor relevance of the past, that is itself a form of exploration and of re-thinking of usual decisions. The best results have been collected for $\epsilon = 1$ and $\alpha = 0.999$. Since over 33% of times a random spot price is applied and every 300 hours the external environment significantly changes, to get at 68% about of the optimal amount of resources is quite satisfying. Similarity in results suggests price contingency and variability equally affect players, and that the tuning of neither ϵ nor α is significantly crucial within the scenario.

F. Exploration vs. Exploitation

In the last scenario, we study the exploration vs. exploitation trade-off. As in Scenario A, the IaaS provider applies a *market-driven* strategy, according to Eq. 12. However, differently from Scenario A, the ten SaaS providers now use different values of ϵ, i.e., they have different exploration and exploitation trade-offs. While player 1 performs no exploration, so that he always takes the *greedy* decision, player 10 takes only random actions, with a full exploration. Table X reports the value of ϵ for all the players. The greater ϵ, the greater the exploration.

Table X
SCENARIO F: VALUES OF ϵ

	SaaS Provider									
	1	2	3	4	5	6	7	8	9	10
ϵ	0	0.05	0.1	0.15	0.2	0.3	0.4	0.5	0.7	1

Table XI reports the percentage of times the players took the five different possible actions.

Table XI
SCENARIO F: SAAS PROVIDER BEHAVIOR

	σ				
SaaS Provider	0.1$	0.2$	0.3$	0.4$	0.5$
1	0.1334	0.3333	0.4666	0.0334	0.03345
2	0.1681	0.5149	0.2649	0.0420	0.01015
3	0.1700	0.4999	0.2598	0.0501	0.02015
4	0.1718	0.4830	0.2566	0.0586	0.02995
5	0.1734	0.4666	0.2533	0.0667	0.03995
6	0.1773	0.4331	0.2465	0.0833	0.05985
7	0.1795	0.3992	0.2405	0.1008	0.08005
8	0.1835	0.3666	0.2334	0.1167	0.09985
9	0.1898	0.3000	0.2207	0.1499	0.13955
10	0.1994	0.2002	0.1994	0.2006	0.20045

Table XII presents the percentage of times the IaaS provider selected one of the five options as $\hat{\sigma}$.

Table XII
SCENARIO F: IAAS PROVIDER BEHAVIOR

0.1$	0.2$	0.3$	0.4$	0.5$
0.169017	0.543107	0.25872	0.029141	0.000015

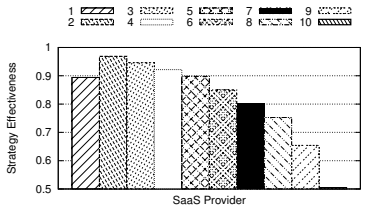

Figure 6. Scenario F: Strategy effectiveness (average ratio $\frac{s_u^*}{s_u^o}$)

As Fig. 6 shows, a little presence of exploration helps the player to behave better and to achieve more resources. This is mostly due to the fact that, thanks to exploration, a player can learn that an action he considered the best actually was not. On the other hand, to explore means to select an option at random; therefore, ϵ should be kept small. In this case, the best performance was obtained with $\epsilon = 0.05$.

VII. Conclusions

In this paper we have addressed the problem of a set of SaaS providers that compete for VM resources provided by an IaaS provider in order to offer Web services with specific QoS constraints. We have proposed a two-stage approach: in the first one, the SaaS providers acquire flat and on demand VMs at a fixed price, while in the second stage they bid and compete to buy on-spot VMs. While the first stage has been solved in [21] through standard optimization techniques, in this paper we have modeled the second stage competition by means of N-armed bandit problems, and studied different techniques and approaches.

We have analyzed the proposed strategies under various scenarios. Our experimental results demonstrate the ability of the SaaS providers to refine their behavior round by round and to determine the best bid in order to maximize their revenue and achieve as many spot resources as possible. We have addressed the importance of a trade-off between exploration and exploitation, considered the role of a discount exponent, and simulated different policies by the IaaS provider, with market-driven, constant, random, N-armed bandit based, and high varying spot prices.

In future work we will study how bidding policies change considering different states based on SaaS users arrival rate, study forward-looking reinforcement learning methods, run simulations with real traces and assess the UCB algorithm [22] as an alternative to the ϵ-greedy heuristic.

References

[1] Amazon, "AWS EC2 pricing," 2013. [Online]. Available: http://aws.amazon.com/ec2/pricing/

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge Univ. Press, 1998.

[3] N. H. Mastronarde and M. van der Schaar, "Automated bidding for media services at the edge of a content delivery network," *IEEE Trans. on Multimedia*, vol. 11, no. 3, 2009.

[4] K. Song, Y. Yao, and L. Golubchik, "Improving the revenue, efficiency and reliability in data center spot market: A truthful mechanism," in *Proc. of IEEE MASCOTS 2013*, 2013.

[5] W. Wang, B. Liang, and B. Li, "Revenue maximization with dynamic auctions in IaaS cloud markets," in *Proc. of IWQoS 2013*, 2013.

[6] M. P. Wellman and J. Hu, "Conjectural equilibrium in multi-agent learning," *Mach. Learn.*, vol. 33, no. 2-3, pp. 179–200, Dec. 1998.

[7] N. H. Mastronarde, V. Patel, J. Xu, and M. van der Schaar, "To relay or not to relay: Learning relaying strategies in cellular device-to-device networks," *CoRR*, 2013, abs/1308.3185.

[8] F. Fu and M. van der Schaar, "Learning to compete for resources in wireless stochastic games," *IEEE Trans. Vehicular Technology*, vol. 58, no. 4, pp. 1904–1919, 2009.

[9] M. Couceiro, P. Romano, and L. Rodrigues, "Polycert: Polymorphic self-optimizing replication for in-memory transactional grids," in *Proc. of 12th ACM/IFIP/USENIX Int'l Conf. on Middleware*, 2011, pp. 309–328.

[10] P. Romano and M. Leonetti, "Self-tuning batching in total order broadcast protocols via analytical modelling and reinforcement learning," in *Proc. of 12th IEEE Int'l Conf. on Computing, Networking and Communications*, 2012.

[11] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing Amazon EC2 spot instance pricing," *ACM Trans. Econ. Comput.*, vol. 1, no. 3, Sep. 2013.

[12] Q. Zhang, E. Gürses, R. Boutaba, and J. Xiao, "Dynamic resource allocation for spot markets in clouds," in *Proc. of USENIX Hot-ICE 2011*, 2011.

[13] J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya, "Tradeoffs between profit and customer satisfaction for service provisioning in the cloud," in *Proc. of 20th Int'l Symp. on High Performance Distributed Computing*, 2011, pp. 229–238.

[14] H. Robbins, "Some aspects of the sequential design of experiments," *Bull. of American Mathematical Society*, vol. 58, no. 5, pp. 527–535, 1952.

[15] T. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in Appl. Mathematics*, vol. 6, no. 1, pp. 4–22, 1985.

[16] D. Kalathil, N. Nayyar, and R. Jain, "Decentralized learning for multi-player multi-armed bandits," in *Proc. of IEEE 51st Conf. on Decision and Control*, 2012, pp. 3960–3965.

[17] M. Babaioff, Y. Sharma, and A. Slivkins, "Characterizing truthful multi-armed bandit mechanisms," *Proc, of ACM EC 2009 (2013's version on arXiv)*, 2009-2013.

[18] R. Gonen and E. Pavlov, "An incentive-compatible multi-armed bandit mechanism," in *Proc. of 26th ACM Symposium on Principles of Distributed Computing*, 2007, pp. 362–363.

[19] P. Rusmevichientong and D. P. Williamson, "An adaptive algorithm for selecting profitable keywords for search-based advertising services," in *Proc. of 7th ACM Conf. on Electronic Commerce*, ser. EC '06, 2006, pp. 260–269.

[20] D. Ardagna, B. Panicucci, and M. Passacantando, "Generalized Nash equilibria for the service provisioning problem in cloud systems," *IEEE Trans. Serv. Comput.*, no. 4, pp. 429–442, 2013.

[21] V. Di Valerio, V. Cardellini, and F. Lo Presti, "Optimal pricing and service provisioning strategies in cloud systems: a Stackelberg game approach," in *Proc. of IEEE 6th Int'l Conf. on Cloud Computing*, ser. Cloud '13, 2013, pp. 115–122.

[22] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.