# QoS-Aware Bidding Strategies for VM Spot Instances: A Reinforcement Learning Approach Applied to Periodic Long Running Jobs

Marco Abundo
University of Roma Tor Vergata
marco.abundo@gmail.com

Valerio Di Valerio
La Sapienza University of Roma
divalerio@di.uniroma1.it

Valeria Cardellini and Francesco Lo Presti
University of Roma Tor Vergata
cardellini@ing.uniroma2.it, lopresti@info.uniroma2.it

***Abstract*—In this paper, we consider an application provider that executes simultaneously periodic long running jobs and needs to ensure a minimum throughput to guarantee QoS to its users; the application provider uses virtual machine (VM) resources offered by an IaaS provider. Aim of the periodic jobs is to compute measures on data collected over a specific time frame. We assume that the IaaS provider offers a *pay for only what you use* scheme similar to the Amazon EC2 service, comprising on demand and spot VM instances. The former are sold at a fixed price, while the latter are assigned on the basis of an auction. We focus on the bidding decision process by the application provider and model the bidding problem as a Q-Learning problem, taking into account the workloads, the maximum completion times since jobs start, the last checkpoint, and the past spot prices observed. In Q-Learning, a form of model-free Reinforcement Learning, the player is repeatedly faced with a choice among $N$ different actions, which will determine immediate rewards or costs and will influence future evolutions. Through numerical experiments, we analyze the resulting bidding strategy under different scenarios. Our results show the application provider ability to refine its behavior and to determine the best action so to minimize the average cost per job, also taking into account checkpointing issues and QoS constraints.**

***Keywords*—Cloud computing, Q-Learning, Reinforcement Learning, VM Provisioning, QoS, Spot Resources**

## I. Introduction

Cloud computing is one of the most disruptive and successful phenomena of the last years. Many companies are exploiting commercially available services provided by Infrastructure as a Service (IaaS) providers, that allow to either scale out or in computing capabilities as need changes, through rapid allocation or deallocation of virtual computing and storage resources. IaaS services let users offer fresh applications without the need of managing the underlying infrastructure.

IaaS providers usually make resources available as Virtual Machine (VM) instances to customers, and the pricing is often governed by a pay-as-you-go model with a per-hour granularity and 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 always saturate the IaaS provider capacity, it can sell the spare capacity as *spot* VMs via an auction, as in the case of Amazon's Elastic Cloud Computing (EC2) [1].

An important class of applications that can exploit the different VM purchasing options is represented by periodic, long running jobs. Examples of such applications include SensorClouds for industrial automation, environmental monitoring, and healthcare [2], [3]. For such applications, the Quality of Service (QoS) levels can be expressed in terms of sustainable minimum throughput, since each job periodically needs to compute measures on data collected over a specific time frame.

In this paper, we consider an application provider that executes periodic batch jobs and uses the VM resources offered by an IaaS provider. The application provider makes available its computation with QoS constraints, consisting in a minimum throughput. The spot bidding requires the application provider to specify the maximum price to pay for a spot VM; then, the IaaS provider sets the spot price. When the application provider bid is lower than the spot price, it will get no resource. Otherwise, the application provider will obtain the requested VM, paying the spot price set by the IaaS provider. On the contrary, on demand VMs are always sold at a fixed price.

Since the IaaS provider can revoke spot VMs without notice for price and demand fluctuations, the application provider needs to face this unreliability through fault-tolerance and backup mechanisms. A well-known mechanism is checkpointing [4], which consists in saving the VM state on a reliable storage and resuming it to restart. We focus on the bidding strategy and assume a periodic checkpointing, performed after a fixed time frame of continuative computation.

In this paper, we study the bidding strategy of the application provider under the realistic assumption that, at each hour, it has no knowledge of the behavior of other IaaS users, their number, the future spot price as well as the recent spot price history not directly experienced. Hence, we assume the application provider can only *learn* the bidding strategy through experience over time. To this end, we resort to Reinforcement Learning (RL) techniques [5].

Specifically, we formulate the bidding strategy as a Q-Learning problem [5]. In Q-Learning, the player is repeatedly faced with a choice among $N$ different actions, which in our setting represent possible bids; every time it submits its decision, it receives an immediate cost or a reward, the system state changes, and the taken action continues to have

consequences over future evolution. Such type of problems is particularly suitable to our setting, since it requires no knowledge of price probability distribution and is forward-looking, thus making possible to take maximum completion times and total costs into account. By observing the incurred immediate and following costs that depend on the chosen actions, the application provider learns over time the bidding strategy which minimizes the average job cost, considering workloads, maximum completion time, past spot prices observed and when the last checkpoint occurred, in order not to waste computation. To the best of our knowledge, Q-Learning has not yet been applied to such cloud resource allocation problems, as well as Reinforcement Learning. Related works regarding spot instances provisioning for batch applications did not focus on the learning problem and the full initial uncertainty of spot price transition probability distribution, as we do. This paper also differs from our previous works, where we proposed approaches based on Stackelberg games [6] and N-armed bandit problems [7], considering short terms computations in settings in which one shot mechanisms are needed rather than forward-looking. However, these models are not adequate for long run estimations.

We study the effectiveness of the bidding strategy through simulation experiments under various settings. Specifically, we assess the application provider behavior under an IaaS provider pricing policy based on Amazon EC2 traces, with different checkpoint frequencies and in various scenarios. We compare our strategy with alternative policies based on blind requests of spot and on demand VMs, and with a Markov Decision Process (MDP) approach [8], in which the model is solved offline and price transition probabilities are known a priori. Our results reveal that, by applying Q-Learning, the application provider succeeds, hour by hour, in refining its behavior, while respecting the QoS constraints and minimizing the costs, with a performance better than blind approaches and comparable with the MDP-based policy. We consider EC2 since it is a popular service and offers to its users the option to combine on demand, fixed price, and reliable VMs along with spot, auction-based, and unreliable VMs. However, our approach is quite general and can be conveniently adapted to many other pricing schemes.

The rest of the paper is organized as follows. In Section II we introduce Q-Learning, while in Section III we provide the problem statement and define the system model. We describe our solution to the bidding problem in Section IV. In Section V we analyze through numerical experiments the behavior of the bidding strategies. In Section VI we review some related research efforts. Finally, we draw some conclusions and give hints for future work in Section VII.

## II. Preliminaries: Q-Learning

Q-Learning is a form of model-free temporal difference (TD) reinforcement learning developed by Watkins [5], [9]. It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without a map of the domains or a priori distributions knowledge.

The basis of each Q-Learning problem simply consists of an agent, a set of states $S$, and a set of possible actions per state $A_s$. When, at time $t$, the agent selects an action $a_t$ in state $s_t$, the system moves from state $s_t$ to state $s_{t+1}$ according to some transition probabilities. The execution of action $a_t$ in the specific state $s_t$ provides the agent with a reward (or cost) $r_{t+1}$. In the following we describe the $Q^*$ function and how it is approximated by the $Q$ function.

### A. $Q^*$ Function

A common way to evaluate the set of rewards (or costs) collected by the agent on time is to consider the total discounted reward (or cost), given by $R = \sum_t \gamma^t r_t$. The $\gamma$ term represents the discount factor, which quantifies the importance given to future values and is in the range $[0, 1]$. If $\gamma$ is 0, then only the current value of $r_t$ is considered. If $\gamma$ is 1, then future values will all have the same weight. The total discounted reward can be useful both in the case of finite (i.e., the set of decision epochs in which an action is selected is finite) and infinite horizon.

The agent aims at maximizing its total reward (or minimizing the cost), by learning, decision after decision and transition after transition, the optimal action for each state.

Let $Q^*$ be the function which associates to a state-action pair the expected total future discounted reward (or cost) that the agent will incur applying action $a$ in state $s$ and thereafter following an optimal policy. If the agent had a priori knowledge and knew the values of $Q^*$, then it could select, at time $t$, the optimal action $a^*$ in state $s_t$ as:

$$a^* = argmax_a Q^*(s_t, a) \tag{1}$$

### B. $Q$ Function Approximation and Action Selection

Actually, the agent needs to approximate $Q^*$ through a $Q$ function, based on rewards (or costs) empirically observed.

At any time $t$, there is at least one action $a' = argmax_a Q(s_t, a)$ (the minimum in case of costs rather than rewards): the greedy action. When the agent chooses it, then it *exploits* its experience. On the other hand, when the agent selects one of the non-greedy actions, then we say it is *exploring*: it is improving the estimation $Q(s_t, a_t)$ of the non-greedy action's value. Generally, exploitation is the best way to maximize the expected reward (or minimize the expected cost) on a local time frame, while exploration lets refine the experience information base and react faster to dramatical environment changes; nevertheless, this strongly depends on the system model, the state space, and the action set. Hence, it is clear that how to learn the best actions and still optimize values requires to carefully trade-off between exploitation and exploration. As discussed more extensively in [7], the agent can obtain this trade-off via the $\epsilon$-greedy heuristic [5], with various effects for different settings. This heuristic is governed by the $\epsilon$ parameter, which ranges from 0 (full exploitation) to 1 (full exploration).

### C. $Q$ Function Update

Each time the agent performs an action $a_t$ in state $s_t$ and gets a reward (or a cost) $r_{t+1}$, it updates the estimation of $Q^*$ for that state-action pair:

$$\begin{aligned} Q(s_t, a_t) &= Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big] \\ &= (1-\alpha)Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma max_a Q(s_{t+1}, a)\big] \end{aligned} \tag{2}$$

The $\alpha$ term represents the learning rate with which new information will overwrite the past one. Setting $\alpha$ to 0 prevents the agent from learning, while 1 shortens the historical horizon to one step. According to [5], such update mechanism dramatically simplifies the algorithm analysis: convergence only requires that all pairs continue to be updated.

## III. System Model

We consider an application provider which executes simultaneously periodic long running jobs by taking advantage of VM resources offered by an IaaS provider. For a clearer description, we first introduce the model considering only a single periodic job, executed on a single VM; then, at the end of Section IV we show that the same model also applies to the general case whereby multiple jobs can be executed at the same time using multiple VMs.

### A. VM Pricing Model

The IaaS provider offers to its customers on demand and spot VMs, using a pricing model similar to Amazon's EC2 [1] (we assume the application provider already exploits reserved flat VMs, so that it can request only on demand and spot VMs). On demand VMs are offered at a fixed price $\delta$ per hour, while spot VMs at a price $\sigma$ per hour. However, $\sigma$ is not fixed, but it rather varies over time according to the IaaS provider pricing strategy and the user bids. For the sake of simplicity, we assume the IaaS provider offers only one size of VMs, i.e., all VMs have the same capacity.

### B. Periodic Jobs

We assume that the periodic job is characterized by a fixed workload $W$, corresponding to the actual execution time needed to complete it. The application provider guarantees QoS constraints in terms of a minimum throughput $1/D$, where $D$, $D \geq W$, is the maximum job duration after its execution starts. As in [10], after a job instance has completed, a new one is submitted; job submissions occur at frequency $1/D$. Examples of such applications are represented by Sensor-Clouds [2], [3], infrastructures that virtualize multiple physical sensors as a "virtual sensor". Sensor-Clouds provide a flexible and reconfigurable Cloud platform for next-generation monitoring and community-centric applications. Another example of periodic jobs is provided by a monitoring application for a call center agent [11], which receives audio chats and performs in real time speech recognition and analysis.

Since data must be collected and processed also aggregating statistics for a minimum time $W$, this problem cannot be significantly speeded up; time is needed in place of computing capacity: the job has to run at least for the $W$ workload. Results must be provided within the completion threshold $D$; of course, a job instance can be completed within any time lower than $D$ with no difference.

### C. Checkpoints

Since spot VMs are less reliable than the other kinds of VMs and can be abruptly revoked at any time owing to their dynamic pricing mechanism, the application provider should consider checkpointing strategies so to mitigate the spot unreliability [10], [12], [13]. Checkpointing consists in the periodic storage of the entire state of the VM on a stable and reliable storage layer. In this way, if a VM executing the job is dropped, then the previous VM state can be restored. In this paper, we consider periodic checkpointing, that is checkpoints are taken with a frequency $f_{cp}$ within a period of continuous activity, based on the nature and the granularity of operations and results to checkpoint. For example, if $f_{cp}$ is 0.5 per hour, a checkpoint will occur after 2 consecutive hours of computation. For the sake of simplicity, we assume that the average overhead $O$ for a single $\langle checkpoint, resume \rangle$ pair is fixed, though the actual CPU time could slightly vary. $O$ is independent of $f_{cp}$: the effort to save and restore the VM state is always the same.

### D. Workload Execution and QoS Constraints

Once $W$ and checkpoint frequency have been fixed, it is easy to determine how many checkpoints will be performed ($n_{cp}$) and to add this effort to $W$ to obtain the total workload $W_O$, given by $W_O = W + n_{cp}O$.

QoS is a generic concept, featured by several attributes in different settings. We consider throughput and cost as suitable attributes to define QoS. Since $D$ is the maximum job duration after its computation starts, the application provider guarantees QoS constraints in terms of a fixed throughput $1/D$.

### E. Provisioning Scheme

We assume that every hour the application provider can allocate and deallocate spot or on demand VMs relying on future spot price predictions and considerations about when the next checkpoint will be performed. The IaaS provider implements some auction mechanism for spot VMs. Hence, the application provider competes for spot resources by submitting to the IaaS provider a bid, which defines the maximum per VM price it is willing to pay. Once an application provider submits a bid and gets a spot resource, it cannot change the bid value any more and the price will remain fixed, until it drops or loses the VM. Spot prices are very variable and can be lower or quite higher than the on demand price for the corresponding VM size [14]. As a consequence, to get spot VMs could turn very convenient compared with on demand VMs, or, on the contrary, it could be really expensive. The application provider goal is to minimize its costs, while respecting QoS constraints. Therefore, it has to optimally decide whether to request a spot or on demand VM and, in case of spot VM, which bid to submit. In the next section we present the strategy we propose to undertake such a decision.

## IV. Bidding Approach

We model the bidding problem as a Q-Learning problem, considering the workloads, maximum completion time, last checkpoint, and past spot prices observed. In the following, we describe the key elements of the Q-Learning problem.

### A. State Space

The state space $S$ is the set of all possible states the application provider can go through. Each state $s \in S$ takes into

account all information needed to summarize in a consistent manner the situation that the system is incurring. We model each state as a 4-ple $s = (\eta, \pi, \psi, \xi)$ where:

- $\eta \in [0, \frac{1}{f_{cp}})$ is the amount of computation performed since last checkpoint;
- $\pi \in [\pi_{min}, \pi_{max}]$ is the last spot price registered;
- $\psi \in [0, W_O]$ is the amount of computation time still needed to complete the task. For example, if $W_O = 5h$ and 2 hours have been already executed, $\psi = 3h$;
- $\xi \in [0, D - W_O]$ is the slack time in addition to $\psi$. For example, if still 2 hours of execution are needed and the job must be completed within 4 hours, $\xi = 2h$.

### B. Actions

Given a state $s \in S$, the set of available actions $A(s)$ basically depends on the amount of *net* slack time $\xi - \eta$, defined as the slack time minus the amount of computation performed since the last checkpoint. It can be viewed as a measure of the urgency in finishing a job. When it drops to 0, to ensure the QoS constraint the only possible action $a_{on\_dem}$ is to request an on demand VM: the application provider cannot take the risk of requesting a spot VM and losing the computation because of a spot price increase, since it would result in a violation of the QoS constraint. Conversely, if $\xi - \eta > 0$ the agent can choose to submit or not a bid for a spot VM, depending on the expected spot price and the amount of performed computation from the last checkpoint. If the application provider decides to submit a bid, it has also to determine how much to bid. However, Tang et al. [10] proved it is enough to consider only two choices, that are 0 and the bid for a spot VM equal to the price of an on demand VM, corresponding to the maximum amount the agent is willing to pay (denoted by $a_{max\_bid}$). This is due to the fact the bid is supposed not to influence the final spot price, either when the spot price is market-driven (many players take part into the auction) or not. Moreover, the player actually pays the spot price and not its bid. Thus, if on the basis of the last spot price observed, last checkpoint and maximum completion time, the player wants to get a spot VM, it is enough to bid as much as a fixed maximum amount.

More formally, given a state $s \in S$, the set of possible actions $A(s)$ can be defined depending on the slack time as follows:

- when there is slack time, i.e., $s$ is s.t. $\xi - \eta > 0$, $A(s) = \{0, a_{max\_bid}\}$. If a spot VM is already running, $a = 0$ corresponds to drop it;
- when the slack time is over, i.e., $s$ is s.t. $\xi - \eta = 0$, $A(s) = \{a_{on\_dem}\}$.

To minimize the average job cost while ensuring the minimum throughput constraint, the agent has to carefully plan its bidding strategy.

### C. Costs

For the application provider, it is critical to complete the job within $D$ since the job computation started avoiding violations of this QoS constraint. The focus of the optimization strategy can be thus put on the VM cost minimization. Therefore, the application provider goal is to minimize the average cost per job on the long time, given by the sum of the single costs $r_t$ paid till the job completion.

When a single action $a_t$ consists in requiring an on demand VM, then the application provider will get the resource and pay the fixed price $\delta$, and one hour of computation will be executed; obviously, such computation could still be wasted if no checkpoint will be performed in the meanwhile. On the contrary, if $a_t$ consists in requiring a spot VM and submitting a bid, then the future evolution will depend on the spot price $\hat{\sigma}_t$ set by the IaaS provider, i.e., the auctioneer.

Let $a_t$ a single bid submitted by the application provider:

- If $a_t < \hat{\sigma}_t$, then no spot VM will be obtained. This could be yet profitable, and the application provider could intentionally lose the opportunity to get a VM. It happens when $\hat{\sigma}_t$ is likely to be high or not convenient;
- if $a_t \geq \hat{\sigma}_t$, then the VM will be obtained, at the price of $\hat{\sigma}_t$ instead of $a_t$. One hour of computation will be executed, but computation could still be wasted if the VM is terminated before a checkpoint.

### D. Action Selection

The bidding mechanism exploits Q-Learning and associates to each state-action pair a value function $Q$, representing an approximation of the future costs the application provider will incur till the job completion. The application provider goal is to minimize the cost per job: each hour, given the state $s$, the best action will be the $a'$ s.t.

$$a' = argmin_a\{Q(s_t, a)\} \tag{3}$$

As explained in Section II, exploitation consists in selecting $a'$, while exploration in applying another action independently on the $Q$ minimization argument.

### E. State Transitions and Q Function Update

Let $\omega_t = 1$ if, through action $a_t$ in state $s_t$ at time $t$, a VM has been obtained, 0 otherwise. Let $\hat{\sigma}_t$ be the spot price set at time $t$ by the IaaS provider. State $s_{t+1}$ is given by:

$$\begin{cases} \eta_{t+1} = \omega_t(\eta_t + 1) \bmod f_{cp} \\ \pi_{t+1} = \hat{\sigma}_t \\ \psi_{t+1} = \psi_t - \omega_t + \eta_t(1 - \omega_t) = \psi_t - \eta_t\omega_t + \eta_t - \omega_t \\ \xi_{t+1} = \xi_t - (\eta_t + 1)(1 - \omega_t) = \xi_t + \omega_t\eta_t - \eta_t + \omega_t - 1 \end{cases} \tag{4}$$

The $Q$ function will be then updated according to Eq. 2.

### F. Parallel Loads

Although we introduced the model for a single periodic job and a single VM, we note that it can be easily extended to deal with parallelism. Indeed, multiple periodic jobs with same or different $W$ and $D$ can be executed, still leveraging on a common $Q$ function; the state space dimension is given by the maximum possible $\eta, \psi, \xi$ and some states are never explored for certain jobs. Moreover, a number $N_{VM} \geq 1$ of VMs exploited at the same time can be considered: the action selection criterion does not vary, costs are proportional to $N_{VM}$, times, state transitions and $Q$ function are the same.

### G. Algorithm Overhead

The proposed Q-learning bidding algorithm can be efficiently implemented. To choose the action, when there is no margin (*i.e.*, when $\xi - \eta = 0$) there is no selection since the user just buys an on demand VM; on the other hand, when $\xi - \eta > 0$ the user bids for one spot instance and only needs to compare $Q(s_t, a_{max\_bid})$ with $Q(s_t, 0)$ to find out whether it is better either to bid the maximum value ($a = a_{max\_bid}$) or to not bid at all ($a = 0$). The state and the Q-functions are updated via simple algebra according to Equations (4) and (2). The memory required to store the $Q$-function is $O(|S| \times |A|)$, which is reasonably small given the time granularity we consider and does not depend on the number of jobs and VMs. Finally, the update time is $O(1)$ since only one entry is updated at each time step.

## V. Experimental Results

In this section we investigate through numerical experiments the behavior of the Q-Learning based bidding strategy. We analyze our strategy against a trace-driven IaaS provider pricing policy, based on spot instance prices of Amazon EC2. We measure the effectiveness of the proposed bidding strategy (named **Q-L** in the following figures) by comparing its achieved costs with the costs incurred by always blindly bidding the $100\%, 85\%$, and $75\%$ of the maximum spot price and paying the current spot price (named **Spot100**, **Spot85**, **Spot75**, respectively), and by applying a precautionary policy (named **On demand**), which always requests on demand VMs at a fixed price without participating to the spot auction. We also compare the Q-L strategy against an offline policy computed by solving a suitable Markov Decision Process (MDP) [8] where the underlying Markov process transition matrix, which captures the price fluctuations over time, is estimated using past price history; such a comparison makes it possible to assess Q-L accuracy as well. The MDP can be solved via standard techniques; the solution is in general time-consuming and not suitable for an online implementation, especially in a non stationary environment where the policy should be recomputed as conditions change over time. As an additional performance metric, we also analyze the job average completion time.

We consider an application provider that applies the Q-Learning bidding strategy, and assume that it executes on a VM a periodic job, characterized by a fixed workload $W$ to be completed within a time period of length $D$. Periodic job instances are executed one by one, and a new instance is submitted after the previous one completed. The overhead for a $\langle checkpoint, resume \rangle$ pair is given by $O = 15min$. It is worth highlighting that, since on demand VMs are used to ensure timely completion in case of no slack, QoS constraints are always met (as described in Sec. IV-B). In all the considered scenarios, in which 500.000 jobs are executed, we set the same parameters for Q-Learning: $\alpha = 0.1$, $\gamma = 1$, and $\epsilon = 0$. The last setting corresponds to no exploration: since actions for spot VMs are only two and with completely different effects, exploration does not perform better. Values for $\alpha$, $\gamma$, $\epsilon$ were chosen in order to have a good trade-off between learning effectiveness and convergence speed, as demonstrated by results we do not include in this paper for space reasons. Higher values of $\alpha$ give faster adaptation by assigning a lower weight to history; $\gamma < 1$ discounts new values, but is usually less appropriate for finite horizons; a lower $\epsilon$ determines convergence speed. The spot prices set by the IaaS provider are randomly generated according to price transition probabilities obtained through a real Amazon EC2 trace, whose values refer to historical data collected from 10/01/2012 to 06/30/2014 for a us-east-m1.large instance [15] and are in the range $[0.016\$, 5.5\$]$; thus, results are based on spot prices given by a dynamic, large scale competition. Furthermore, the cost of on demand VMs per hour is equal to $0.175\$$ in all scenarios, *i.e.*, $a_{max\_bid} = 0.175\$$. This value corresponds to the price of an Amazon EC2 us-east-m1.large instance [16] and is quite lower than spot price spikes. The player considers 5 price ranges for the last spot price registered: $(0, 0.025\$]$, $(0.025\$, 0.05\$]$, $(0.05\$, 0.1\$]$, $(0.1\$, 1.0\$]$, $(1.0\$, 5.5\$]$. Experiments only focus on a single job since the same holds when multiple parallel jobs are executed, except that convergence of the algorithms is faster.

First, we study the performance of the bidding policy for different deadlines (**Scenario A**). We set $W = 4h$ and $f_{cp} = 1/2h$, while we consider different values for $D$: $6h$, $9h$, $12h$.

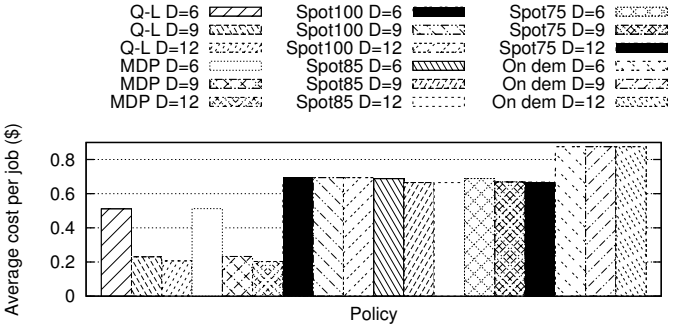


Figure 1. Scenario A: Average total cost per job

Figure 1 shows the effectiveness of the bidding strategy in terms of average total cost per job. We observe that the average costs achieved by Q-L are analogous to those of MDP and this confirms the effectiveness of the Q-Learning based strategy. In some cases, Q-L is even more convenient than MDP: while the latter is computed offline according to an approximation of spot prices based on the price ranges, the former considers the actual values collected at runtime. Costs decrease as $D$ increases, due to a larger slack and a more relaxed deadline. In addition, costs with Q-L are always far lower than those incurred with the blind policies. Compared with the Spot100 policy, we observe a reduction over $35\%$ for $D = 6h$, about $200\%$ for $D = 9h$, about $235\%$ for $D = 12h$; values are similar for the Spot85 and Spot75 policies. Given the On demand policy, results point out costs over $70\%$ lower for $D = 6h$, about $280\%$ lower for $D = 9h$, about $325\%$ lower for $D = 12h$. Thus, the higher $D$, the more the advantage over blind policies, since the application provider has more opportunities (*i.e.*, more time) to take advantage of the price fluctuations and wait for lower spot prices.

Table I shows the average completion times for all policies, when $D$ varies. When $D$ is higher, completion times increase accordingly.

Figure 2 shows the fraction of times the different actions

Table I. SCENARIO A: AVERAGE COMPLETION TIME PER JOB (H)

| | $D$ | | |
|---|---|---|---|
| Policy | $6h$ | $9h$ | $12h$ |
| Q-Learning | 4.7133 | 5.6927 | 6.5523 |
| MDP | 4.7368 | 5.1832 | 5.4237 |
| Spot100 | 4.500 | 4.500 | 4.500 |
| Spot85 | 4.501 | 4.503 | 4.503 |
| Spot75 | 4.502 | 4.505 | 4.506 |
| On demand | 4.500 | 4.500 | 4.500 |

Table II. SCENARIO B: AVERAGE COMPLETION TIME PER JOB (H)

| | $f_{cp}$ | | |
|---|---|---|---|
| Policy | $1/h$ | $1/2h$ | $1/3h$ |
| Q-Learning | 8.6325 | 8.2451 | 7.6416 |
| MDP | 8.2785 | 7.9456 | 7.4933 |
| Spot100 | 7.750 | 6.750 | 6.500 |
| Spot85 | 7.753 | 6.754 | 6.504 |
| Spot75 | 7.754 | 6.754 | 6.507 |
| On demand | 7.750 | 6.750 | 6.500 |

are chosen under Q-L. When $D$ is low, due to the strict time budget, the more likely actions are $a_{max\_bid}$ (max bid) and $a_{on\_dem}$ (on demand). As $D$ increases, the fraction of times action 0 (corresponding to no bid) is chosen increases, while the provider decreases the number of times it resorts to on demand instances. The frequency of the spot action maximum bid increases at first, when it is preferred to the on demand action (from $D = 6h$ to $D = 9h$), but then it diminishes as the no bid action becomes more convenient due to the looser deadline (from $D = 9h$ to $D = 12h$).

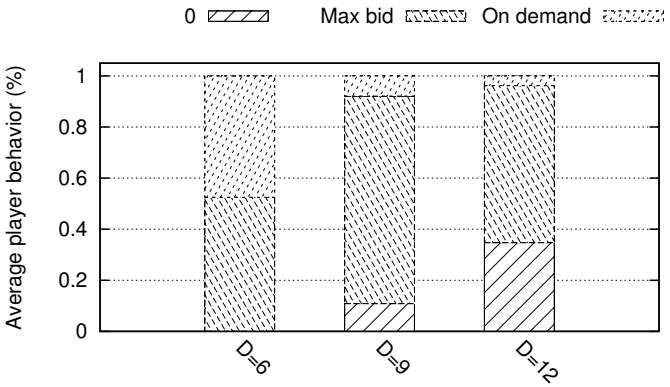


Figure 2. Scenario A: Fraction of times actions no bid (0), max bid, on demand are taken

Finally, Fig. 3 shows the convergence of the Q-Learning average cost for the different deadlines.

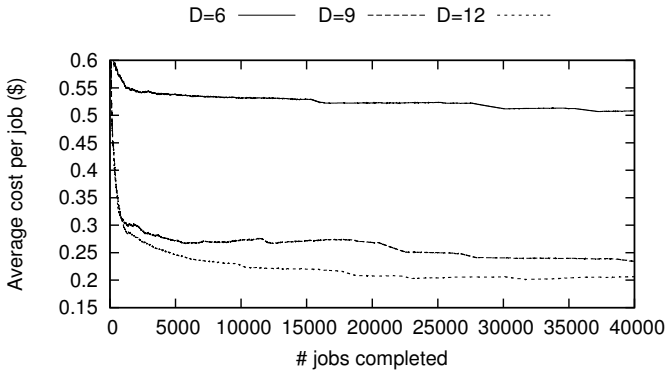


Figure 3. Scenario A: Q-Learning convergence

We now turn our attention to checkpoint frequencies (**Scenario B**): given $W = 6h$ and $D = 10h$, we consider different checkpoint frequency values $f_{cp}$: $1/1h$, $1/2h$, $1/3h$. The overhead for a $\langle checkpoint, resume \rangle$ pair is fixed and equal to $O = 15min$. Figure 4 shows the effectiveness of the bidding strategy in terms of average total cost per job. As in Scenario A, the average costs obtained online by Q-L are analogous to those of MDP, which is based on an a priori knowledge. The lower costs obtained via Q-L (and MDP as well) are those with $f_{cp} = 1/2h$. Indeed, when $f_{cp} = 1/h$, owing to the checkpoint-resume overhead $O = 15min$, the total workload $W_O$ grows, and more computation is required. On the other hand, when $f_{cp} = 1/3h$, although the overall checkpoint overhead is lower, the application provider is more likely to have to repeat already executed computations that were not checkpointed. In all cases, costs are quite lower than those given by the Spot and the On demand policies. Compared with the Spot100, Spot85 and Spot75 policies, we observe a reduction over $155\%$ for $f_{cp} = 1/1h$ and $f_{cp} = 1/2h$, and over $140\%$ for $f_{cp} = 1/3h$. Given the On demand policy, results reveal costs over $220\%$ lower for $f_{cp} = 1/1h$ and $f_{cp} = 1/2h$, and about $200\%$ lower for $f_{cp} = 1/3h$. Thus, Q-L is more advantageous when checkpoints are performed every hour and every 2 hours, while, when $f_{cp} = 1/3h$ its effectiveness is lower, although still significant, because of time spanning between two checkpoints.

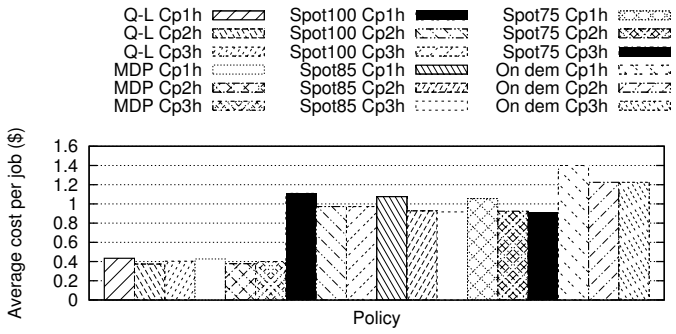


Figure 4. Scenario B: Average total cost per job

Table II reports the average completion times for the bidding strategies at different checkpoint frequencies. As expected, the completion times are lower when checkpointing is executed less frequently.

Figure 5 shows the fraction of times each action was chosen. In this scenario, due to the reduced slack $D - W_O$, in order to meet the QoS constraint, the provider is more likely to request an on demand instance or to bid a price equal to that of an on demand VM than in other scenarios. The minimum cost is incurred for $f_{cp} = 1/2h$, corresponding to the case where the fraction of time the bid is 0 attains its maximum.

Figure 6 shows the Q-L average cost convergence for the three checkpoint frequencies.

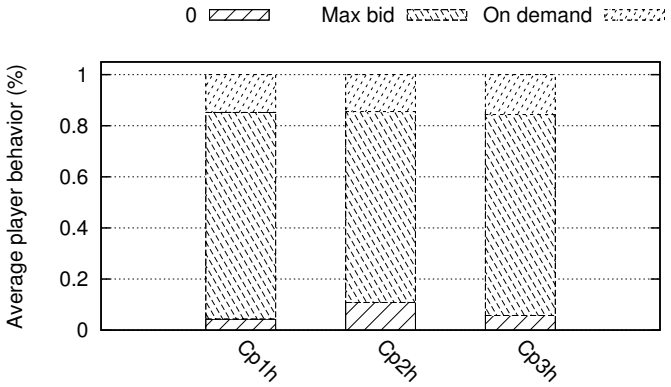


Figure 5. Scenario B: Fraction of times actions no bid (0), max bid, on demand are taken

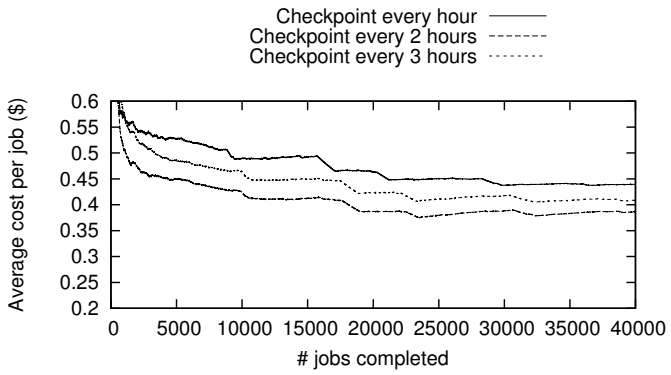


Figure 6. Scenario B: Q-L convergence

As final scenario, we consider different values of the workload (**Scenario C**) with $W = 4h$, $7h$, $10h$, fixed $D = 14h$ and $f_{cp} = 1/2h$. As Figures 7, 8, and 9 and Table III show, when $W$ increases, the slack is reduced and the behavior is similar to the previously presented scenario, where $W$ is fixed and $D$ is instead reduced.

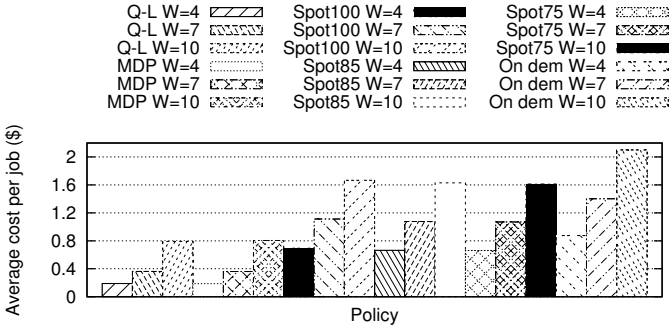


Figure 7. Scenario C: Average total cost per job

## VI. Related Work

A considerable number of research efforts have recently focused on spot instances provisioning and pricing.

Tang et al. [10] proposed a set of bidding strategies, formulating the problem as a Constrained Markov Decision Process (CMDP), but differently from us they considered constraints on the average completion time. As in our paper, they considered a dual-option spot strategy: bidding the maximum spot price or bidding zero dollars (give up). Nevertheless, in our paper we bid the on demand price, that can be much lower than the maximum spot price. This leads to a different model and solution technique. Moreover, we apply Q-Learning and thus transition probabilities between states are not necessary.

Table III. Scenario C: Average completion time per job (h)

| | $W$ | | |
|---|---|---|---|
| Policy | $4h$ | $7h$ | $10h$ |
| Q-Learning | 9.4491 | 9.5874 | 11.9923 |
| MDP | 5.4524 | 9.3889 | 11.9775 |
| Spot100 | 4.500 | 7.750 | 11.250 |
| Spot85 | 4.505 | 7.758 | 11.255 |
| Spot75 | 4.508 | 7.776 | 11.257 |
| On demand | 4.500 | 7.750 | 11.250 |

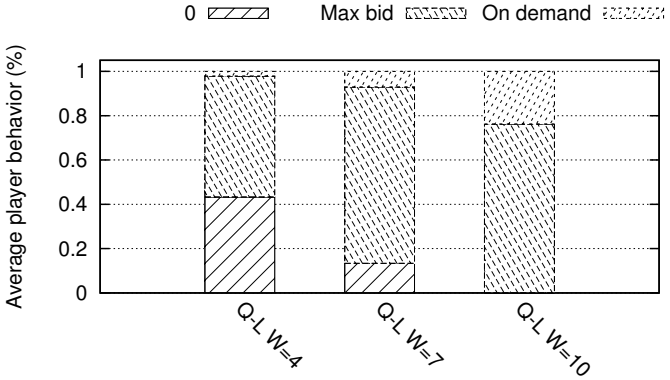


Figure 8. Scenario C: Fraction of times actions no bid (0), max bid, on demand are taken

Song et al. [17] investigated the problem of designing a bidding strategy from a Cloud service broker perspective, applying a profit-aware dynamic bidding. However, they did not consider checkpointing and their model requires spot price history data in order to reconstruct the semi-Markovian chain at the basis of the profit maximization problem.

Menache et al. [18] proposed a machine learning-based algorithm, addressing the trade-off between cost and performance, and between on demand and spot VMs usage. They did not consider checkpointing, while we do; moreover, the same policy is used for an entire job, and values are updated only when the job ends. Finally, the bid for spot VMs is either fixed or set as the weighted average of past spot prices plus a safety parameter, i.e., it is not given by an optimization problem.

Poola et al. [19] presented a job scheduling algorithm on Cloud resources, using, as in our paper, two different pricing models (spot and on demand) and aiming at reducing the execution cost whilst meeting the workflow deadline. Differently from our work, a bid failure probability is considered, based on one month prior to the start of the execution and the spot prices until the point of estimation are used. Moreover, bid values increase gradually with the workflow execution as the deadline becomes closer.

Zafer et al. [20], as in our paper, modeled a job as a fixed computation request with a deadline constraint and designed a

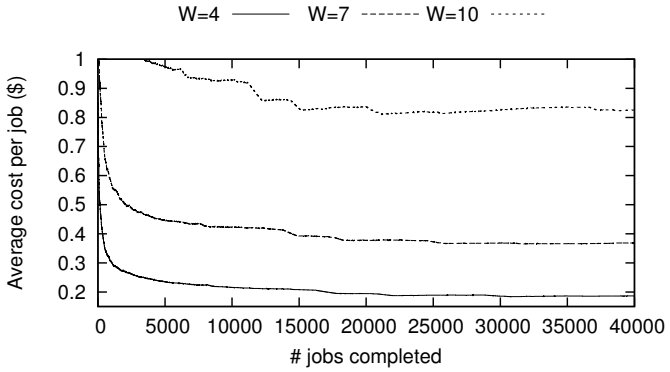


Figure 9. Scenario C: Q-L convergence

bidding policy minimizing the average cost per job. In contrast with our work, they obtained analytical and closed-form results under a Markov spot price evolution, i.e., their policy needs the spot price history and a training phase. Moreover, they assumed that at the slot boundaries the VM image can be stored and reloaded for subsequent computations utilizing Cloud storage, with no checkpointing.

Yi et al. [13] studied how checkpointing can be used to minimize cost and volatility of resource provisioning and compared various policies. They focused more on checkpointing and less on the bidding strategy than our work.

Chohan et al [21] described how to improve the runtime of MapReduce workflows by using Amazon EC2 spot instances; however, they neither consider deadline constraints nor checkpointing. Moreover, they studied the job performance as function of a constant bid using real traces, hence there is no learning of an optimal dynamic user strategy at runtime.

Voorsluys and Buyya [12] presented a resource allocation strategy that addresses the problem of running compute-intensive jobs on a pool of intermittent VMs, and also employs price and runtime estimation mechanisms, as well as fault-tolerance techniques. As in our work, the maximum spot bid is set to the on demand price. Differently, Reinforcement Learning is not exploited, and the spot price history is fed to the bidding strategy before it starts.

Q-Learning has been widely used in the literature [22], [23], [24], [25], [26], [27] but, to the best of our knowledge, it has never been applied to Cloud resource allocation.

## VII. Conclusions

In this paper we addressed the problem of an application provider that executes batch jobs and needs to ensure a minimum throughput to guarantee QoS levels to its users. To this end, the provider uses the spot and on demand VMs offered by an IaaS provider. We proposed a bidding strategy based on a Q-learning approach focusing on workload, maximum completion time, last checkpoint and past observed spot prices. We addressed the fulfillment of a minimum throughput as crucial QoS constraint, and evaluated the proposed bidding strategy under different scenarios and real spot price traces. Our experimental results prove the ability of the application provider to refine its behavior round by round and to determine the best action in order to maximize its revenues and minimize the average cost per job.

In future work we will study how bidding policies change when considering variable workloads and completion constraints, longer scenarios, more providers, different checkpointing techniques, and alternative SLA and cost models, in which a degree of urgency can be specified for a job. Furthermore, we plan to explore robust MDP techniques along with reinforcement learning to increase the robustness of our approach to abrupt environment changes.

## Acknowledgements

This work is partially supported by the European ICT COST Action IC1304 Autonomous Control for a Reliable Internet of Services (ACROSS).

## References

[1] Amazon, "AWS EC2 pricing," 2015. [Online]. Available: http://aws.amazon.com/ec2/pricing/

[2] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing," in *Proc. of 13th Int'l Conf. on Network-Based Information Systems*, ser. NBiS'10, 2010.

[3] A. Alamri, W. Ansari, M. Hassan *et al.*, "A survey on sensor-cloud: architecture, applications, and approaches," *Int'l J. of Distributed Sensor Networks*, 2013.

[4] S. Yi, A. Andrzejak, and D. Kondo, "Monetary cost-aware checkpointing and migration on Amazon cloud spot instances," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, 2012.

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

[6] 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.

[7] M. Abundo, V. Di Valerio, V. Cardellini, and F. Lo Presti, "Bidding strategies in QoS-aware cloud systems based on N-armed bandit problems," in *Proc. of IEEE 3rd Symp. on Network Cloud Computing and Applications*, ser. NCCA'14, 2014, pp. 38–45.

[8] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic, Dynamic Programming*. Wiley, 1994.

[9] C. J. Watkins, "Learning from delayed rewards," *PhD Thesis, University of Cambridge*, 1989.

[10] S. Tang, J. Yuan, C. Wang, and X.-Y. Li, "A framework for Amazon EC2 bidding strategy under SLA constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 2–11, 2014.

[11] G. Mishne, D. Carmel, R. Hoory *et al.*, "Automatic analysis of call-center conversations," in *Proc. of 14th ACM Int'l Conf. on Information and Knowledge Management*, ser. CIKM '05, 2005, pp. 453–459.

[12] 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*, ser. AINA'12, 2012, pp. 542–549.

[13] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of spot instances via checkpointing in the Amazon Elastic Compute Cloud," in *Proc. of IEEE CLOUD 2010*, 2010, pp. 236–243.

[14] B. Javadi, R. K. Thulasiram, and R. Buyya, "Characterizing spot price dynamics in public cloud environments," *Future Gener. Comput. Syst.*, vol. 29, no. 4, pp. 988–999, 2013.

[15] B. Javadi, R. Thulasiram, and R. Buyya, "Amazon public cloud (EC2) spot price archive," 2015. [Online]. Available: http://spot.scem.uws.edu.au/ec2si/

[16] M. Khristo, "EC2 on-demand vs reserved instance cost savings calculator," 2013. [Online]. Available: http://mikekhristo.com/ec2-ondemand-vs-reserved-instance-savings-calculator/

[17] Y. Song, M. Zafer, and K.-W. Lee, "Optimal bidding in spot instance market," in *Proc. of IEEE INFOCOM 2012*, 2012.

[18] I. Menache, O. Shamir, and N. Jain, "On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud," in *Proc. of 11th Int'l Conf. on Autonomic Computing*, ser. ICAC'14, Jun. 2014, pp. 177–187.

[19] D. Poola, K. Ramamohanarao, and R. Buyya, "Fault-tolerant workflow scheduling using spot instances on clouds," *Procedia Computer Science*, vol. 29, pp. 523–533, 2014.

[20] M. Zafer, Y. Song, and K.-W. Lee, "Optimal bids for spot VMs in a cloud for deadline constrained jobs," in *Proc. of IEEE CLOUD 2012*, 2012.

[21] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: Using spot instances for MapReduce workflows," in *Proc. of 2nd USENIX Conf. on Hot Topics in Cloud Computing*, ser. HotCloud'10, 2010.

[22] A. Barto and S. Singh, "On the computational economics of reinforcement learning," in *Connectionist Models: Proc. of 1990 Summer School*, 1990.

[23] R. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proc. of 7th Int'l Conf. on Machine Learning*, 1990, pp. 216–224.

[24] D. Chapman and L. P. Kaelbling, "Input generalization in delayed reinforcement learning: An algorithm and performance comparisons," in *Proc. of Int'l Joint Conf. on Artificial Intelligence*, ser. IJCAI'91, 1991.

[25] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," in *Proc. of 9th National Conf. on Artificial Intelligence*, vol. 2, 1991, pp. 768–773.

[26] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[27] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1-2, pp. 81–138, 1995.