BLMDP: A new bi-level Markov decision process approach to joint bidding and task-scheduling in cloud spot market

Mona Naghdehforoushha\textsuperscript{1}, Mehdi Dehghan Takht Fooladi\textsuperscript{2}, Mohammad Hossein Rezvani\textsuperscript{1,∗}, Mohammad Mehdi Gilanian Sadeghi\textsuperscript{1}

\textsuperscript{1}Department of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran
\textsuperscript{2}Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran

Received: 17.08.2021 • Accepted/Published Online: 15.12.2021 • Final Version: 31.05.2022

Abstract: In the cloud computing market (CCM), computing services are traded between cloud providers and consumers in the form of the computing capacity of virtual machines (VMs). The Amazon spot market is one of the most well-known markets in which the surplus capacity of data centers is auctioned off in the form of VMs at relatively low prices. For each submitted task, the user can offer a price that is higher than the current price. However, uncertainty in the market environment confronts the user with challenges such as the variable price of VMs and the variable number of users. An appropriate strategy should both maximize the user’s utility and determine the best bid for him/her. In this paper, we aim to solve the problem of joint minimizing the cost of processing tasks and maximizing user satisfaction. Our proposed method, which we call bi-level Markov decision-making process (BLMDP), works on two levels. At the top level, it selects the most appropriate user bids and adjusts the spot price to minimize the cost of VMs on the cloud provider side. At a low level, it decides to admit tasks in such a way as to maximize the user side satisfaction. Performance evaluation based on real data collected from Amazon web services shows that BLMDP manages to minimize cloud provider’s costs and maximize user gain more effectively compared to heuristic methods.

Key words: Cloud computing, bidding strategy, spot price, task scheduling, Markov decision process (MDP), Amazon EC2

1. Introduction

Many cloud computing providers offer their computing capabilities to end-users through the cloud computing markets (CCMs) \cite{1}. For example, Amazon EC2 \cite{2,3} provides virtual machines (VMs) with three pricing methods: on-demand, reserved, and spot \cite{3,4}. Among these, the spot market is the most important market, which works based on the auction mechanism \cite{5}. First, a price is set for each spot virtual machine (SVM), and then each user tests his/her chances of renting resources by sending a bid to the cloud server. The spot price is usually updated on an hourly basis \cite{1,6}.

So far, various research has been conducted on bid prediction for the users in CCMs \cite{6–9}. In summary, the most important weaknesses of the previous research are as follows:

1) Some methods have used unrealistic assumptions such as the absence of a deadline. An assumption that does not exist in any of the current applications of the internet.

∗Correspondence: rezvani@qiau.ac.ir

This work is licensed under a Creative Commons Attribution 4.0 International License.
2) Most previous methods have focused solely on one goal, which is mainly to minimize user costs [1, 10, 11]. It seems that the problem has a multiobjective nature. A method that is compatible with real-world requirements must both consider user satisfaction and be able to adapt to spot price fluctuations.

3) The previous methods do not take into account the uncertainty of other users in determining a user’s bid for the future time slot.

4) To increase his/her profit, a user may perform some tasks earlier than others, so that the deadline is met. Once the service is assigned to the user, he/she may schedule tasks on a preferred basis to maximize time usage.

Unfortunately, the previous methods do not pay attention to this at all. It is necessary to solve the bidding problem for SVMs in combination with the above problems. Of course, some previous research has addressed the issue of scheduling, but in isolation [4, 12]. Task scheduling can be done in a two-step strategy: a) designing an acceptance/rejection policy that specifies which tasks must be queued for execution, b) designing an order policy that specifies how tasks should be performed.

Reinforcement learning, especially Markov decision process (MDP) [13–16], is one of the most common tools for predictive optimization. To address this problem, we jointly solve the following two optimization problems using MDP:

a) At the top level, the subproblem of user bidding is solved, which aims to reduce the cost paid to the provider.

b) At the low level, the subproblem of optimizing the task scheduling is solved, which aims to increase user satisfaction and increase the total gain of performing the tasks for him/her.

If the bidding is successful, the requested SVMs will be awarded to the user. Now, the user can schedule tasks according to a predetermined deadline. It is enough for the user to pay a cost equivalent to the current spot price to the cloud provider instead of the bid price.

The proposed approach considers both issues simultaneously in a two-level framework. The top-level focuses on bidding at one-hour intervals to receive service, while the low level focuses on admitting/blocking tasks by considering their deadlines. Note that most CCMs, such as Amazon, update their prices at one-hour intervals. In this paper, to model the above two subproblems, a reinforcement learning method called bi-level Markov decision process (BLMDP) is used. It is an analytical model in which decisions are made through interactions between the two levels. Here, a bottom-up hierarchy is considered. This means that low-level decisions also affect high-level decisions. To complete each service, the lower level is given enough time to be able to maximize profits.

The rest of the paper is organized as follows: Section 2 reviews the literature, Section 3 formulates the problem in terms of two MDP subproblems and presents distributed algorithms, Section 4 evaluates the performance of the proposed model. Finally, Section 5 concludes the paper and outlines trends of future research.

2. Related works
So far, extensive research has been conducted on CCMs. Some researchers [26] have examined the issue of cost balancing and the availability of SVM to users. Some of these studies [27] have proposed algorithms to optimize monetary costs to solve the problem of minimizing execution costs. In some other works [28], risk reduction and rental costs have been explored. A review of the literature reveals that there are two major directions in CCM research: a) examining pricing mechanisms and providing resources on the cloud provider side, b) designing
new strategies for bidding on the user side. So far, extensive efforts have been made to discover the pricing mechanism on the provider side. For example, some studies have used double auction mechanisms. Given that the subject of this paper is the design of a user-side strategy, we do not analyze provider-side research in-depth.

The choice of a bidding strategy usually depends on the application requirements, the adopted fault tolerance mechanism, availability, monetary constraints, and task deadline. In general, the bidding policy should be designed in such a way that by changing the price of VMs over time, the user can use them when the price is relatively low and, thus, reduce the cost of execution. Therefore, a tradeoff must be made between reliability and cost. In older research, bidding policies were designed based on a combination of reserved/on-demand prices and spot price forecasting. Recently, the use of static methods has become obsolete, and dynamic methods have been used instead. Interested readers can refer to to study the most important dynamic bidding schemes. Given that our proposed method falls into the category of dynamic ones, we focus on explaining them.

We now proceed to describe dynamic methods based on reinforcement learning. The key assumption here is that when a cloud provider changes the spot price per hour, users must also change their bids. Kaminski and Szufel proposed an adaptive strategy to minimize the cost of performing tasks. This is done according to the spot price of Amazon EC2 in a certain geography, the frequency of requesting VM instances, and the bid of other users. Liu et al. developed a hidden Markov model (HMM) to describe the different demand markets for cloud computing and the degree of volatility of spot prices. Their results showed that the proposed model can predict the spot price more accurately than the regression-based forecasting methods. This model can successfully make predictions of less than 5 h. Gari et al. proposed an approach to obtain a comparative budget allocation policy using the Markov decision process (MDP). Their method can reallocate the budget at each stage of the workflow. Their results showed that the proposed method leads to cost savings and cost reduction compared to baseline methods. In a similar study, Cai et al. proposed a dynamic ARIMA-based prediction method and a Markov-based model. Another study using the reinforcement learning approach was conducted by Cong et al. They first designed a new model for predicting user-perceived value. The plan is based on the relationship between user personality, service price, service quality (QoS), user satisfaction, and perceived value in the cloud service market. Based on the forecasting model, they then designed an MDP-based cloud pricing mechanism that can learn service pricing decisions to jointly maximize profits and minimize costs. Their results show that the efficiency of the proposed pricing mechanism is almost 20% better than the baseline methods. In a similar study, Xie and Liu used Q-learning to deduce the optimal price from historical trading data. They first designed a discrete-time dynamic pricing scheme and formulated an MDP to describe the evolving dynamics of price-dependent demand. Their results show that the proposed dynamic model can lead to an increase in revenue of up to 20% over fixed pricing. A similar study was conducted by Tang et al. using the constrained MDP method and considering the deadline for tasks. They compared the proposed dynamic method with a static baseline one and showed that it performs better in the case of uncertainty. Interested readers can refer to to study more about reinforcement learning methods in this topic.

Unlike previous research, our proposed method works on two levels. It offers several advantages to the user. First, it allows the user to make informed decisions about the bid price, which, in turn, affects risk and cost. Second, the proposed method combines low-level bid price information, gain factor, and job execution time to find the best task schedule when renting an SVM. The core idea behind combining the two sub-problems is
to avoid hasty decisions that may incur costs. Achieving such an optimal strategy can postpone the execution of less important tasks. It also ensures that tasks are executed within their predetermined deadlines. Third, the proposed method uses the checkpointing technique to store intermediate results of the task. Therefore, the results of the task are not lost, which, in turn, leads to greater reliability.

It is worth noting that spot bidding strategies are not limited to cloud networks. It has been used in other engineering areas, especially electric microgrids. Examples of recent research include crude oil price forecasting [22] and renewable energy resource behavior forecasting [23].

3. Proposed model
The proposed method works hierarchically at two levels, one with a long period and the other with a shorter period. Each level has a state space, an action space, a state transition probability function, and a gain function (instant cost). Each state-action pair in the top-level determines the state transition probability and the gain function for the lower level. In other words, the value of a low-level policy affects a higher-level decision. To this end, we consider the following two subproblems:

1) Process modeling at a top-level, i.e., the sub-problem of deciding on the bid price to obtain the service.
2) Process modeling at a low-level, i.e., the sub-problem of task scheduling for the user.

We model each of the above two subproblems using the reinforcement learning approach with the MDP and provide an algorithm for each. Finally, two optimization sub-problems are solved jointly using the BLMDP method. The notations used in this paper are shown in Table 1.

3.1. Top-level MDP Modeling
With any change in the spot price, the requesting user must submit his/her bid price. Therefore, at the beginning of each time slot (each hour), the user must decide whether or not to bid. We denote the set of possible states and the set of actions by \(X\) and \(A\), respectively. At each hour, the next hour action is selected according to the policy \(\mu\). Our goal in this sub-problem is to find the optimal policy for user bidding. We now proceed to formulate the components of the MDP model for this sub-problem. Readers interested in studying MDP in more depth can refer to [13, 30].

A) States: Let us denote the system state at time slot \(h\) by \(x_h = [\text{Spotprice}_h, \text{Decision}_{h-1}, \text{Decision}_h]\), where \(h = \{h_0, h_0 + 1, ..., H\}\). Here, \(h_0\) and \(H\) denote the current and final epochs, respectively. Also, \(\text{Spotprice}_h\) denotes the spot price in the current time \(h\). Similarly, \(\text{Decision}_{h-1}\) and \(\text{Decision}_h\) represent the result of the bid decision in the previous and current time slot, respectively. We represent all system states with \(X = \{x_1, x_2, ..., x_n\}\), which is a finite and countable set.

B) Actions: Let us denote the set of actions per hour with a pair \(A = \{\text{bidding, give up}\}\). Here, \text{bidding} and \text{give up} show the user bidding and canceling the auction, respectively. At each moment, the move to the next state is calculated based on the price transition matrix and the price history of the cloud provider. Therefore, for the next hour, the user must either decide to bid the maximum price or give up and perform the task checkpointing.

As seen in Figure 1, based on the decision made at the current time slot, the user may be moved to one of four states.

According to Figure 2, depending on the action performed in the previous, current, and next consecutive times, the execution progress for a computational task can be calculated as follows:
Table 1. Mathematical notations.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h = (h_0, h_0 + 1, \ldots, H)$</td>
<td>The set of decision epochs</td>
</tr>
<tr>
<td>$H$</td>
<td>The horizon</td>
</tr>
<tr>
<td>$x_h$</td>
<td>The integrated state of the time slot $h$ in bidding subproblem</td>
</tr>
<tr>
<td>$a_h$</td>
<td>The action of the time slot $h$ in bidding subproblem</td>
</tr>
<tr>
<td>$X$</td>
<td>The set of all states in bidding subproblem</td>
</tr>
<tr>
<td>$S$</td>
<td>The set of all states in task scheduling subproblem</td>
</tr>
<tr>
<td>$A$</td>
<td>The set of all actions in the bidding subproblem</td>
</tr>
<tr>
<td>$V$</td>
<td>The set of all actions in the task scheduling subproblem</td>
</tr>
<tr>
<td>$W$</td>
<td>The total volume of work</td>
</tr>
<tr>
<td>$t_{\text{check}}$</td>
<td>The checkpointing time</td>
</tr>
<tr>
<td>$t_{\text{resume}}$</td>
<td>The resuming time</td>
</tr>
<tr>
<td>$P_{h,i}(a_h)$</td>
<td>The state transition probability function</td>
</tr>
<tr>
<td>$\Lambda(\sigma, \tau)$</td>
<td>The probability of transition between prices</td>
</tr>
<tr>
<td>$\delta_x(x_h)$</td>
<td>The function of changes in other states relative to a specific state in the bidding subproblem</td>
</tr>
<tr>
<td>$\text{Exe}(x_h, a_h)$</td>
<td>The execution progress</td>
</tr>
<tr>
<td>$E[\text{Exe}]$</td>
<td>The Expected execution progress</td>
</tr>
<tr>
<td>$E[\text{Cost}]$</td>
<td>The Expected monetary cost to finish the entire task</td>
</tr>
<tr>
<td>$\text{Cost}(x_h)$</td>
<td>The price paid at the time slot $h$</td>
</tr>
<tr>
<td>$\rho(x_h, a_h)$</td>
<td>The occupation measure of a state and bid option</td>
</tr>
<tr>
<td>$t_{\text{deadline}}$</td>
<td>The deadline</td>
</tr>
<tr>
<td>$\mu$</td>
<td>The optimal bidding policy</td>
</tr>
<tr>
<td>$m$</td>
<td>The type of the task based on processing time</td>
</tr>
<tr>
<td>$M$</td>
<td>The number of different types of arrived tasks</td>
</tr>
<tr>
<td>$\beta(m)$</td>
<td>The average time to keep a task in the queue in task scheduling subproblem</td>
</tr>
<tr>
<td>$\alpha(m)$</td>
<td>The interval rate of a task of type $m$</td>
</tr>
<tr>
<td>$L$</td>
<td>The length of the processing queue</td>
</tr>
<tr>
<td>$s_0$</td>
<td>The state without considering the event in the task scheduling sub-problem</td>
</tr>
<tr>
<td>$S_0$</td>
<td>The set of all states without considering the event in the task scheduling subproblem</td>
</tr>
<tr>
<td>$e(m)$</td>
<td>Task event of type $m$</td>
</tr>
<tr>
<td>$e$</td>
<td>Each event in the task scheduling subproblem</td>
</tr>
<tr>
<td>$E$</td>
<td>The set of all possible events</td>
</tr>
<tr>
<td>$r(s_0)$</td>
<td>The sum of the rates of different events in a state in the task scheduling subproblem</td>
</tr>
<tr>
<td>$r$</td>
<td>The maximum sum of the rates of different events in the task scheduling subproblem</td>
</tr>
<tr>
<td>$\text{reward}(m)$</td>
<td>Instant reward function in task scheduling subproblem</td>
</tr>
<tr>
<td>$\text{penalty}(m)$</td>
<td>Instant penalty function in task scheduling subproblem</td>
</tr>
<tr>
<td>$G_s(v)$</td>
<td>Instant gain function in task scheduling subproblem</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>The discount factor in the Belman equation</td>
</tr>
<tr>
<td>$V(s)$</td>
<td>Value function</td>
</tr>
<tr>
<td>$\pi^*$</td>
<td>The optimal policy for the task scheduling subproblem</td>
</tr>
<tr>
<td>$\Pi$</td>
<td>The set of all policies at the top-level of the BLMDP</td>
</tr>
<tr>
<td>$T_1$</td>
<td>The length of time interval in the top-level of the BLMDP</td>
</tr>
<tr>
<td>$T_2$</td>
<td>The length of time interval in the low-level of the BLMDP</td>
</tr>
<tr>
<td>$V(s_0)$</td>
<td>The value function in the low-level of the BLMDP</td>
</tr>
<tr>
<td>$V^*$</td>
<td>The optimal value of the value function</td>
</tr>
<tr>
<td>$Q_{s', v}(s)$</td>
<td>The probability of transition from state $s$ to state $s'$ by action $v$</td>
</tr>
<tr>
<td>$G(s, v, s')$</td>
<td>Instant gain function in the low-level of BLMDP</td>
</tr>
<tr>
<td>$R_V(x_h, a_h, \pi^*_L)$</td>
<td>The top-level gain function in the BLMDP</td>
</tr>
</tbody>
</table>
\[ Exe(x_h, a_h) = \begin{cases} 
1 - t_{\text{check}} & \text{if } (x_h = [\text{Spotprice}_h, \text{Decision}_{h-1} = \text{bidding}, \text{Decision}_h = \text{bidding}], a_h = \text{give up}) \\
1 - t_{\text{resume}} & \text{if } (x_h = [\text{Spotprice}_h, \text{Decision}_{h-1} = \text{give up}, \text{Decision}_h = \text{bidding}], a_h = \text{bidding}) \\
1 - t_{\text{check}} - t_{\text{resume}} & \text{if } (x_h = [\text{Spotprice}_h, \text{Decision}_{h-1} = \text{give up}, \text{Decision}_h = \text{bidding}], a_h = \text{give up}) \\
1 & \text{if } (x_h = [\text{Spotprice}_h, \text{Decision}_{h-1} = \text{bidding}, \text{Decision}_h = \text{bidding}], a_h = \text{bidding}) \\
0 & \text{if } \text{otherwise} 
\end{cases} \]
where \( t_{\text{check}} \) and \( t_{\text{resume}} \) are the time required to checkpoint (inspect) and resume the current task, respectively. Therefore, at some hours, depending on the circumstances, execution progress may be reduced by \( t_{\text{check}} \) or \( t_{\text{resume}} \) or both.

**C) Transition Probability:** This function is defined by moving from \( x_i \) to \( x_j \). When the system is in a state \( x_i \), the probability of going to the state \( x_j \) by acting \( a_i \) is \( P_{i,j}(a_i) = \Lambda(\sigma, \tau) \). Otherwise, the probability is zero, i.e., \( P_{i,j}(a_i) = 0 \). Here, \( \Lambda(\sigma, \tau) \) is the probability of going from the state in which the price is \( \sigma \) to a state in which the price is \( \tau \).

**D) Objective Function:** The objective function, \( E[\text{Cost}] \), is the mathematical expectation of the cost paid by the user to the cloud provider during the execution of a task. This function is expressed as follows:

\[
\min E[\text{Cost}] = \sum_{a_p \in A} \sum_{a_p \in A} \rho(x_h, a_h) \times \text{Cost}(x_h) \times t_{\text{deadline}}
\]  

(2)

, where:

\[
\text{Cost}(x_h) = \begin{cases} 
\text{Spotprice}_h & \text{if } \text{Decision}_h = \text{bidding} \\
0 & \text{if } \text{Decision}_h = \text{give up}
\end{cases}
\]  

(3)

In the above expression, \( t_{\text{deadline}} \) denotes the deadline, i.e., the maximum time allowed to perform the task. Also, \( \rho(x_h, a_h) \) is the probability of occurrence for a pair \((x_h, a_h)\) which is determined by the decision policy.

To clarify the matter, we illustrate an exemplary execution scenario in Figure 3. Suppose a user wants to run a task with two VM instances in parallel. If this task requires 14 hours to be completed, the workload in each VM instance is 7 hours. The total time available for execution is 9 hours from which 7 hours are effective considering that time of checkpointing is 0.5 hours and time of resume is 0.5 hours. Given that the spot price fluctuates during execution and the bid price is US$ 0.4. Hence, it goes out of bid for time intervals [5,6,9,11]. Altogether, the components of the total cost consist of a) 3 hour of running at US$ 0.1 b) 4 hours of running at US$ 0.2, and c) 2 of running at US$ 0.3. So, the total cost is US$ 1.7. It is concluded that with two VM instances, the effective cost per instance is US$ 1.7/2=0.85. The average effective execution is 7/9=0.78 per hour.

**Figure 3.** An exemplary execution scenario.
E) **Constraints:** Our top-level MDP problem has the following four constraints:

\[ E[Exe] \geq \frac{W}{t_{\text{deadline}}} \]  

(4)

\[ \rho(x_h, a_h) \geq 0 \quad \forall x_h, \forall a_h \]  

(5)

\[ \sum_{x_h \in X} \sum_{a_h \in A} \rho(x_h, a_h) = 1 \]  

(6)

\[ \forall x_h \in X : \sum_{x_h \in X} \sum_{a_h \in A} \rho(x_h, a_h) \times (\delta_{x_i}(x_i) - P_{i,j}(a_i)) = 0 \]  

(7)

The constraint in (4) guarantees that if the mathematical expectation of the task is not less than \( \frac{W}{t_{\text{deadline}}} \), then it will be completed before the deadline. Here, it is assumed that the processing capacity is one unit per hour, and the workload is \( W \). Also, \( E[Exe] \) denotes the mathematical expectation of the amount of task performed per hour. The constraints in (5)-(6) guarantee that \( \rho(x_h, a_h) \) is a random variable. The constraint in (7) ensures that, in each state, the rate equilibrium law is established. Here, the rate at which the system moves to the state \( x_i \) when it is currently in the state \( x_j \) is defined as follows:

\[ \delta_{x_j}(x_i) = \begin{cases} 1 & \text{if } (i = j) \\ x & \text{otherwise} \end{cases} \]  

(8)

The linear programming (LP) problem, which is defined in (2)-(8) can be solved by known methods such as SIMPLEX. By solving this problem, the desired occurrence index can be obtained in the form of a \((x_h, a_h)\) pair. We now proceed to obtain the probability of selecting each action \( a_h \) in each state \( x_h \). Let us denote this optimal value by \( \mu(a_h \mid x_h) \). *Algorithm 1* shows the above routine. With the task deadline \( t_{\text{deadline}} \) and the transition probability matrix \( P \), we can simply obtain \( \mu(a_h \mid x_h) \).

**Algorithm 1 Pseudocode of optimal bidding strategy.**

**Input:** deadline \( t_{\text{deadline}} \), transition probability matrix \( P \)

**Output:** optimal bidding strategy \( \mu \)

Solve the problem of (2)-(8) to get the updated occurrence index \( \rho(x_h, a_h), \forall x_h \in X, \forall a_h \in A \);

Calculate optimal bidding strategy \( \mu(a_h \mid x_h) = \frac{\rho(x_h, a_h)}{\sum_{a_h \in A} \rho(x_h, a_h)} \)

We now proceed to achieve the optimal decision policy to perform the user task. As shown in *Algorithm 2*, having the optimal strategy of the previous step as input, it is possible to determine which action to be selected for the next hour in each state. Finally, the length of each time slot, \( T_1 \), can be measured. This value is passed as input to the low-level optimization subproblem, which will be described below.
Algorithm 2 Pseudocode for optimal decision-making policy $\Pi$.

**Input:** optimal bidding strategy $\mu$ (computed from Algorithm 1), current state $x_h \in X$

**Output:** the bidding decision for the next hour

Randomly choose a bid option based on bidding strategy $\mu$ calculated from Algorithm 1 for the next hour:

```plaintext
if bidding strategy for next hour == "give up" then
  set a checkpoint at the end of the current hour.
end if
```

Perform bidding decision correspondingly in the next hour;

3.2. Low-level MDP modeling

We assume that there are $M$ different types of tasks in the system. Every $m$-type task, which $m \in \{1, 2, ..., M\}$, takes $\beta(m)$ time to be processed, and its arrival time is $\alpha(m)$. Therefore, the service rate of each $m$-type task is $\frac{1}{\beta(m)}$. The decision to accept/reject each task depends on the number of tasks in the processing queue. If the task is not accepted, a fine penalty $\delta(m)$ is given to the user and otherwise a reward $\pi(m)$. Let denote the number of states and actions of this subsystem by $S$ and $V$, respectively. Also, we assume $X \cap S = \emptyset$ and $A \cap V = \emptyset$. The user goal is to maximize the amount of the reward and penalty difference in the long run.

**A) States:** The queue state of each $m$-type task is defined as $s_0 = (s_0(1), ..., s_0(M))$, where $1 \leq m \leq M$.

Therefore, the set of states of this subsystem can be represented as follows:

$$S_0 = \{ s_0 \in \mathbb{R}^M \mid \sum_{m=1}^{M} s_0(m) \times \beta(m) \leq L, s_0(m) \in \{0, 1, 2, ...\}\}$$

(9)

We define three types of events for each $m$-type task. If a similar task is entered into the system, then $e(m) = 1$. If a same-type task goes out, then $e(m) = -1$. Also, a value of $e(m) = 0$ is a sign of no change. The set of all possible events is defined with a tuple $e = (e(1), ..., e(M))$ as follows:

$$E = \{ e \mid e \in \{-1, 0, 1\}^M, \sum_{m=1}^{M} |e(m)| \leq 1\}$$

(10)

Each state of this subsystem is itself defined as a pair $(s_0, e)$ including queue and event status. Finally, the state space is as follows:

$$S = \{ s = (s_0, e) \mid s_0 \in S_0, e \in E, e(m) \geq 0 \text{ if } s_0(m) = 0\}$$

(11)

**B) Actions:** Control action involves deciding whether or not to allow a task to enter the process queue. Action space is simply defined as follows:

$$V = \{ v_{\text{accept}}, v_{\text{reject}} \}$$

(12)

**C) Transition Probability:** Let $Q_{ss'}(v)$ denote the probability of transition from state $s = (s_0, e)$ to state $s' = (s'_0, e')$ when the action $v$ is selected. We now use a technique called uniformity to approximate the problem from continuous-time to discrete-time. To do this, we first define a parameter called $r$ equal to the maximum total rate of events in each state:

$$r = \max_{s_0 \in S_0} \{ \sum_{m=1}^{M} (\alpha(m) + s_0(m)\beta(m))\}$$

(13)
Also, for each $s_0 \in S_0, r(s_0)$ is defined as follows:

$$r = \sum_{m=1}^{M} (\alpha(m) + s_0(m)\beta(m)) \tag{14}$$

The probability that a $m$-type task enters the processing queue is $\frac{\alpha(m)}{r}$. The probability that an $m$-type task will be out of the processing queue is $\frac{s_0(m)\beta(m)}{r}$. Also, the probability of not changing the processing queue is $1 - \frac{r(s_0)}{r}$. Now, the transition probability $Q_{ss'}(v)$ can be calculated in different states:

The probability that a task arrives and is accepted by the user is:

$$Q_{ss'}(v) = \begin{cases} \frac{\alpha(m)}{s_0(m)\beta(m)} & \text{if } s'_0 = s_0 + e, \text{ and } e'_m = 1 \\ \frac{s'_0(m)\beta(m)}{r} & \text{if } s'_0 = s_0 + e, \text{ and } e'_m = -1 \\ 1 - \frac{r(s'_0)}{r} & \text{if } s'_0 = s_0 + e, \text{ and } e'_m = e^* \end{cases} \tag{15}$$

The probability that a task arrives and is not accepted by the user is:

$$Q_{ss'}(v) = \begin{cases} \frac{\alpha(m)}{s_0(m)\beta(m)} & \text{if } s'_0 = s_0, \text{ and } e'_m = 1 \\ \frac{s'_0(m)\beta(m)}{r} & \text{if } s'_0 = s_0, \text{ and } e'_m = -1 \\ 1 - \frac{r(s'_0)}{r} & \text{if } s'_0 = s_0, \text{ and } e'_m = e^* \end{cases} \tag{16}$$

The probability that a task is removed from the queue is:

$$Q_{ss'}(v) = \begin{cases} \frac{\alpha(m)}{s_0(m)\beta(m)} & \text{if } s'_0 = s_0 + e, \text{ and } e'_m = 1 \\ \frac{s'_0(m)\beta(m)}{r} & \text{if } s'_0 = s_0 + e, \text{ and } e'_m = -1 \\ 1 - \frac{r(s'_0)}{r} & \text{if } s'_0 = s_0 + e, \text{ and } e'_m = e^* \end{cases} \tag{17}$$

If there is no change in the system, we have:

$$Q_{ss'}(v) = \begin{cases} \frac{\alpha(m)}{s_0(m)\beta(m)} & \text{if } s'_0 = s_0, \text{ and } e'_m = 1 \\ \frac{s'_0(m)\beta(m)}{r} & \text{if } s'_0 = s_0, \text{ and } e'_m = -1 \\ 1 - \frac{r(s'_0)}{r} & \text{if } s'_0 = s_0, \text{ and } e'_m = e^* \end{cases} \tag{18}$$

In other cases, we have $Q_{ss'}(v) = 0$.

**D) Objective Function:** For each state, we define an objective function. A value $reward(m)$ is awarded when an $m$-type task is accepted to enter the queue in the state $s = (s_0, e) \in S$. Conversely, if it is not accepted for admission to the queue or is accepted but the queue is full, then it will be fined $penalty(m)$. So, we write:

$$G_{s}(v) = \begin{cases} reward(m) & \text{if } V = v_{accept}, e_m = 1, s_0 + e \in s_0 \\ -penalty(m) & \text{if } V = v_{reject} \text{ or } V = v_{accept}, e_m = 1, s_0 + e \notin s_0 \end{cases} \tag{19}$$

To calculate the optimal scheduling policy, it is sufficient to solve the above-mentioned MDP by the value iteration method (VIM). As shown in Algorithm 3, the parameter $\theta$ is used to check convergence to the optimal solution. Also, the discount factor $0 < \lambda < 1$ serves to apply the effect of the value of previous states in calculating the value of the new state.
Algorithm 3 Pseudocode of VIM for computing optimal policy $\pi$

**Input:** the set of all states $S$, set of all actions $V$, state transition function $Q_{ss'}(v)$, the reward function $G_s(v)$, threshold $\theta$, discounted factor $\lambda$.

**Output:** optimal policy $\pi(s)$, Value function $V(s)$

$V_0[i] = 0, K = 0$;

Repeat $k = k + 1$;

ForEach $s \in S$ do

$V_k[s] = \max_v \sum_{s'} Q(s' | s, v)(G(s, v, s') + \lambda V_{K-1}[s'])$;

Until $\forall [v_k[s] - V_{K-1}[s]] < \theta$;

ForEach $s \in S$ do

$\pi[s] = \arg \max_v \sum_{s'} Q(s' | s, v)(G(s, v, s') + \lambda V_{K-1}[s'])$;

Return $\pi, V_K$;

Line 5 in Algorithm 3 shows the Bellman equation, which can be solved by many common LP methods. The time complexity of each iteration is of order $O(|S|^2|V|)$. Here each state contains several queues, each with several tasks of different types. Due to the great variety of the number of permutations, solving the above equation is very time-consuming. So, we proceed to solve it with a reduced infinite method in which the value function is defined as follows [13]:

$$U_\pi(s_0) = E\left\{\sum_{t=0}^{\infty} \lambda^t G(s, v, s') | \pi, s_0\right\}$$

In the above equation, $s_0$ indicates the initial state. The optimal value of the value function is $V^*(s) = \max_\pi V_\pi(s)$, and the optimal policy is equal to $\pi^* = \arg \max_\pi V_\pi(s)$.

3.3. Joint optimization of two subproblems using BLMDP

As described in Section 3.1, the top-level MDP model is first solved to optimize user bidding and average cost. Then, at the beginning of each interval $T_i$, a low-level MDP model is solved to optimize the gain of executing tasks by scheduling them. In the other words, the low-level MDP is called at the beginning of each time slot $h$ with the state $x_h$, the action $a_h$, and the objective function $E\{Exe(x_h, a_h)\}$. As shown in Figure 4, based on a decision made at the top-level, the bottom level is allowed to make a policy based on that. This policy is used to achieve the goal defined at this level.

The low-level policy is displayed in the time slot $h$ with $\pi^h \in \Pi$, where $\Pi$ is the set of the low-level MDP policies. At the time $h$, having the state $x_h$ and the action $a_h$, the reduced mathematical expectation function of the gain is defined as follows:

$$V(s_{1h}) = E\left\{\sum_{t=1}^{H} \lambda^t G(s_{t+1}, v_{1h}) | a_h\right\}$$

where $0 < \lambda < 1$ is the discount factor, $s_{1h}$ is the initial state at the low-level, and the $V(s_{1h})$ is the mathematical expectation at state $s_{1h}$. The optimal value of each state is given as follows:

$$V^*(s) = \min_{\pi^h \in \Pi_L} \left\{\sum_{s' \in S} Q(s, v, s')(G(s, v, s') + \lambda V^*(s'))\right\}$$
As mentioned before, after solving the above equation by VIM, the optimal scheduling policy $\Pi^*_t$ can be obtained. At the top-level MDP in any state $s_h$ with any action $a_h$, the Gain function at the time $h$ is defined as follows:

$$R_V(s_h, a_h, \pi^h_L) = E\{V(s_{h+1}) + M(a_h)\}$$  \hspace{1cm} (23)

(23) simply shows the average of the utility from the low-level MDP plus the cost from the top-level MDP, starting from $s_0$. Starting from a different initial state $s_1$, a different $\pi^h_L$ policy is obtained. The two levels of the MDP are related by $a_h$ and $s_1$. The top-level policy is represented by $\pi_v \equiv (a_1, a_2, ..., a_H)$ and the set of all top-level policies is represented by $\Pi_V$. Ultimately, the joint optimization of bidding and task scheduling problems is expressed as follows:

$$\min_{\pi_v \in \Pi_v} \min_{\pi^h_L \in \Pi^*_h} E\{\sum_{h=1}^{H} R_V(s_h, a_h, \pi^h_L)\}$$  \hspace{1cm} (24)

4. Performance evaluation
We have used the Amazon EC2 dataset [3] with different scenarios. This dataset is related to 29 days of Amazon transactions that have been provided to users by the spot pricing approach. Here, the transition probability matrix is calculated based on the frequency of repetition of each price in the first 15 days of this interval. To take into account the data correlation, the Markov price chain is created based on data from the last 14 days.

As a scenario, we use a specific VM instance of the Amazon CCM called US East.m3.2xlarge. Here, the price per hour of use in the form of reserved, on-demand, and spot pricing is $0.3243, 0.532$, and $0.089$, respectively. Figure 5 shows the price changes of the VM instance, from 2015/11/15 to 2016/12/22 for 29 days.

There are three types of tasks, the specifications of which are shown in Table 2. As stated in Section 3.2, when a new task arrives, an appropriate decision is made to admit/block it. The algorithm also determines whether the task deadline can be met. When an -type task is admitted in the processing queue, the user receives an "instant reward". It is determined by the size or importance of the task for the user. For example, if multiple tasks depend on a specific task, it is highlighted as an important one. Conversely, if the task is not admitted, the user receives an "instant penalty". Also, if the task is admitted to enter the queue but there is no empty room,
the user will still receive an "instant penalty". Because such a task cannot be queued in time, it is postponed. This, in turn, may force the user to pay an additional fee to perform the task in the future. In general, if a task fails to be admitted, it is not deleted; instead, it is resubmitted by the user.

The simulation was performed in a 64-bit JRE 10.0.2 environment on a core i7-2670QM computer with 6 GB of memory and a Microsoft Windows 8 operating system. Table 3 shows the details of the settings used in the simulation scenarios. These scenarios will be described later in Section 4.2.

### Table 2. Task specifications in the experimental data-set.

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival rate factor (According to Poisson distribution)</th>
<th>Average waiting time of the task in the queue (minutes)</th>
<th>Instant Gain</th>
<th>Instant Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0.1</td>
<td>6</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>T2</td>
<td>0.2</td>
<td>9</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>T3</td>
<td>0.3</td>
<td>12</td>
<td>0.6</td>
<td>0.6</td>
</tr>
</tbody>
</table>

### Table 3. Settings used in the simulation scenarios.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W )</td>
<td>100 hours</td>
</tr>
<tr>
<td>( t_{\text{deadline}} )</td>
<td>200 hours</td>
</tr>
<tr>
<td>( t_{\text{check}} )</td>
<td>10 minute</td>
</tr>
<tr>
<td>( t_{\text{resume}} )</td>
<td>5 minute</td>
</tr>
<tr>
<td>( L )</td>
<td>10</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>0.97</td>
</tr>
</tbody>
</table>

### 4.1. Description of benchmarks

We compare the performance of the proposed method with two heuristic methods, whose settings are very similar to our one. For comparison, we use the heuristic bidding method in Algorithm 4 [29] and the heuristic
dynamic scheduling method in Algorithm 5 [24]. Both of these methods take into account the deadline and make decisions at any hour. In Algorithm 4, the probability of each price occurrence is calculated based on its repetition on the price history. Then the prices are arranged in ascending order. For each user, the bidding decision is made based on the ratio of the remaining volume of his/her task to the available time. If this ratio is greater than the probability of the first minimum price occurrence, then the user will bid the highest price. Similarly, for other cases that occur with a lower probability than this ratio, the highest price is offered. The first price that meets these conditions is randomly selected. Otherwise, no price bid is offered by the user.

Algorithm 4 Pseudocode of heuristic bidding strategy [29]

**Input:** current state, the reward of scheduling strategy, remained work, the cumulative probability of stored prices CPSP  
**Output:** bid decision  
for $i = 0$ to number of unique prices do  
if $(\frac{\text{remained work}}{\text{current state} - \to \text{available time}} + \text{reward of scheduling strategy}) > CPSP[i]$ then  
    bid maximum price;  
    last price index = $i + 1$;  
end if  
if $(\text{current state} - \to \text{index} == \text{last price})$ then  
    randomly bid maximum price;  
end if  
end for

Algorithm 5 Pseudocode of heuristic task scheduling strategy (inspired by the random early detection algorithm.) [24]

**Input:** current size of queue, arrived task, threshold 1, threshold 2, value (reward of arrived task, average queue size)  
**Output:** admission to arriving task  
if average – queue – size $< \text{Threshold}1$ then  
    add arrived task to the queue;  
else  
    if average – queue – size $> \text{Threshold}2$ then  
        drop arriving task;  
    else  
        if random – value $< \text{value}$ then  
            add task to processing queue;  
        end if  
    end if  
end if

In Algorithm 5, which is designed based on the well-known random early detection (RED) method, a threshold is defined for the average workload rate. If the queue length is less than this threshold, the task enters the processing queue. Otherwise, the task is not added to the queue. For situations between these two thresholds, the algorithm operates randomly. In this case, a function is defined based on the average queue size and the arrived task, and then a random number is generated. If this random number is less than the value generated by the function, the task will be added to the queue.
4.2. Analysis of results

We consider four scenarios for comparing the efficiency of the proposed method (BLMDP) with the heuristic methods of Algorithms 4 and 5 as follows:

**Scenario 1:** In this scenario, we perform joint optimization of the user bid submission subproblem and the task scheduling subproblem using the Bi-Level MDP method (BLMDP).

**Scenario 2:** In this scenario, we solve the user bid submission subproblem using the MDP and the task scheduling subproblem with the heuristic method (MDP-H).

**Scenario 3:** In this scenario, we solve the user bid submission subproblem using the heuristic method and the task scheduling subproblem with the MDP method (H-MDP).

**Scenario 4:** In this scenario, we solve both subproblems of user bid submission and task scheduling with the heuristic method (H-H).

Figure 6 shows the bidding/give-up decisions for different strategies. In other words, this figure shows which of the two "bidding" or "give up" decisions the user has made in each time slot during a 200-hours session. A value of 1 on the vertical axis indicates that the user has made the "bidding" decision. Conversely, a value of zero indicates a "give up" decision by him/her. Given the price changes in Amazon data [3], this figure shows how much the user has spent using each of the four strategies. If in a time slot, his/her decision is "bidding", he/she will pay the relevant VM price to rent it. Otherwise, he/she will not be charged any costs in that time slot. Now to see which strategy is better, we need to add up the total user costs throughout the session. The strategy with the lowest total cost is the best.

![Bid / Give up decisions](image)

**Figure 6.** Maximum/zero bid for the considered four strategy.

Figure 7 shows the total cost for different strategies. As can be seen from the figure, using the BLMDP strategy leads to the lowest cost for the user. Another interesting point in it is that the use of MDP in combination with the heuristic method (MDP-H and H-MDP) results in better performance compared to the case where only the heuristic method is used (H-H). Note that Figure 7 is cumulatively plotted, meaning that for each curve, its cost values are obtained by adding the cost values of the curves below it. The interesting point about this figure is that a similar upward trend is seen in all four strategies.

Figure 8 shows the percentage of cost reduction in the proposed method compared to the heuristic one. Figure 8a shows the effect of checkpointing time and Figure 8b shows the effect of the ratio of workload to...
the deadline. Recall from (1) that when checkpointing time increases, the mathematical expectation of task completion time decreases. In other words, when the checkpointing time is longer, a larger volume of the entire task remains to be completed. The efficiency of the proposed method is more remarkable than the heuristic method according to Figure 8a. In other words, the horizontal axis in Figure 8a shows the value of \( 1 - t_{\text{check}} - t_{\text{resume}} \), which is simply written as \( 1 - (t_{\text{check}} + t_{\text{resume}}) \). Note that when the value in parentheses increases, the total value of the expression, \( 1 - (t_{\text{check}} + t_{\text{resume}}) \), decreases. This means that only a small portion of the task has been done and most of it remains to be done. It can be seen that the cost reduction of the proposed method is much greater than that of the heuristic method.

Figure 8. The percentage of cost reduction in the proposed method compared to the heuristic method for a) checkpointing time, b) workload to deadline ratio.
As shown in Figure 8b, when the workload-to-deadline ratio is not high (for example at the 20% point), the cost of the proposed method is significantly reduced compared to the heuristic method. This means that when the workload-to-deadline ratio is low, the efficiency of the proposed method is significant compared to the heuristic method. It is observed that by increasing this ratio, the efficiency of the heuristic method decreases.

Figure 9 shows a comparison of the gains earned from scheduled tasks between the proposed method and the heuristic method. Note that in this figure, the values of gain (the vertical axis) are plotted on a logarithmic scale. Recall from (9)-(11) that one of the most important parameters affecting the system state is the queue size. Then in (21)-(23), the queue size resulting from the scheduling operation, causes a direct impact on user gain. Also, as the processing queue size increases, the gain value increases in both methods. Obviously, the larger the processing queue size, the higher the percentage of tasks transferred from the input queue to the processing queue. Hence, it is very unlikely that a task will be admitted in the queue, but there is no empty room in the queue. This results in fewer tasks being penalized due to the short queue. Therefore, it is quite natural that the gain function increases in any case. However, the gain value of the BLMDP method is always higher than that of the heuristic method.

Now, to get a deeper insight into the methods, we compare them using statistical moments.

![Figure 9](image-url) Comparison of the benefits of scheduled tasks between the proposed method and the heuristic method.

As shown in Figure 10, the mean, variance, and standard deviation of cost in the proposed strategy are lower than those of other strategies. Regarding the average cost analysis, sufficient explanations were provided in the previous charts. The low standard deviation of cost in the proposed strategy indicates that it has fewer cost fluctuations than others. This provides an attractive implication for CCM designers. It intuitively shows that the proposed method is more stable than others.

5. Conclusion and future trends

In this paper, we targeted one of the critical issues in cloud services, namely user bid prediction. With the aim of joint optimizing the bidding strategy and scheduling tasks, we presented a method using reinforcement learning. This method solves the two sub-problems of high-level bidding optimization and low-level scheduling...
optimization jointly using Markov decision process (MDP) theory. Performance evaluation on Amazon EC2 data showed that the proposed method could not only reduces the cost of renting for the cloud provider but also increases the satisfaction of performing tasks on the user side compared to heuristic methods.

One of the future research trends is the use of other reinforcement learning methods such as deep RL. Examining other on-demand scheduling policies may also have a positive effect on checkpointing and resuming costs. Another line of future research could be to predict task execution time to make the acceptance/rejection process more accurate.

References


