Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 5080647, 11 pages
https://doi.org/10.1155/2019/5080647

Research Article

A Reinforcement Learning Based Auto-Scaling Approach for SaaS Providers in Dynamic Cloud Environment

Yi Wei[1], **Daniel Kudenko**[2,3], **Shijun Liu**[1], **Li Pan**[1], **Lei Wu**[1] and **Xiangxu Meng**[1]

[1]*School of Software, Shandong University, Jinan, China*
[2]*Department of Computer Science, University of York, York, UK*
[3]*Saint Petersburg National Research Academic University of the Russian Academy of Sciences, St Petersburg, Russia*

Correspondence should be addressed to Shijun Liu; lsj@sdu.edu.cn and Li Pan; panli@sdu.edu.cn

Received 1 August 2018; Accepted 17 January 2019; Published 3 February 2019

Academic Editor: Rafał Stanisławski

Cloud computing is an emerging paradigm which provides a flexible and diversified trading market for Infrastructure-as-a-Service (IaaS) providers, Software-as-a-Service (SaaS) providers, and cloud-based application customers. Taking the perspective of SaaS providers, they offer various SaaS services using rental cloud resources supplied by IaaS providers to their end users. In order to maximize their utility, the best behavioural strategy is to reduce renting expenses as much as possible while providing sufficient processing capacity to meet customer demands. In reality, public IaaS providers such as Amazon offer different types of virtual machine (VM) instances with different pricing models. Moreover, service requests from customers always change as time goes by. In such heterogeneous and changing environments, how to realize application auto-scaling becomes increasingly significant for SaaS providers. In this paper, we first formulate this problem and then propose a Q-learning based self-adaptive renting plan generation approach to help SaaS providers make efficient IaaS facilities adjustment decisions dynamically. Through a series of experiments and simulation, we evaluate the auto-scaling approach under different market conditions and compare it with two other resource allocation strategies. Experimental results show that our approach could automatically generate optimal renting policies for the SaaS provider in the long run.

1. Introduction

With the rapid development of cloud computing technologies such as virtualization and service-oriented architectures, large numbers of agile and scalable cloud-based applications are delivered across the Internet. Their providers, i.e., SaaS providers, take advantage of commercially available cloud facilities provided by IaaS providers to run these applications cost-effectively. From the perspective of SaaS providers, one remarkable benefit of such a market pattern is that they are allowed to rent virtualized resources flexibly from IaaS vendors based on their actual demands. For most SaaS providers, they derive their profits from the margin between the revenue obtained from customers and the expense of renting cloud resources from IaaS providers. Therefore, how to allocate appropriate resources to balance application performance and operating cost is a crucial issue for them.

On the one hand, underprovisioned resources are not capable of bearing customer workloads thereby affecting customer satisfaction; on the other hand, resource overprovisioning can result in idle VM instances and unnecessary cost. Thus, the best behaviour for SaaS provider is to rent as few VM instances as possible while still ensuring applications offer sufficient processing capacity to meet customer needs. Furthermore, because of market diversity, IaaS providers adopt different pricing models for their VM instances. Common mechanisms include on-demand pattern, reserved pattern, and spot pattern [1]. Considered from the point of view of SaaS providers, different choices of VM instances with different pricing patterns will also affect their renting cost. Therefore, SaaS providers need to consider how to automatically generate proper renting policy which takes into account the types, quantity, and pricing patterns of VM instances.

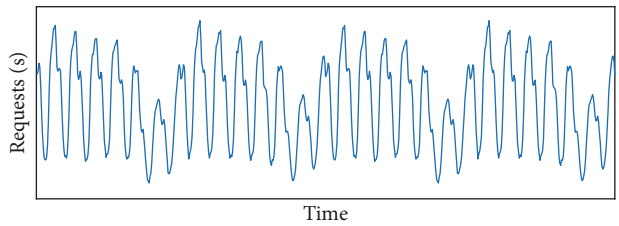


Figure 1: Wikimedia workload between June 1, 2015 and June 28, 2015 (4 weeks) [2].

Since customer workload is always changing, achieving automatic scaling up or down to respond to the dynamic service requests from customers is challenging. In fact, although customer workloads change continually as time goes by, these random workloads still have certain regularity in some degree. Figure 1 shows that Wikimedia workload follows typical time-of-day and day-of-week patterns. Similarly, as for most cloud-based enterprise applications, they receive more requests from customers during working hours in day time and become vacant at night time. If SaaS providers could learn from their previous trading experiences, better renting policies could be generated. Hence, in this work, we aim to design a learning based auto-scaling approach for SaaS providers to help them generate appropriate VM renting policies for their applications in order to adapt to the uncertainty of the market.

The rest of this paper is structured as follows. Section 2 gives an overview of relevant work in this field. In Section 3, we define the three-tire cloud market model with some assumptions and then present the formulation of cloud application auto-scaling problem in this context. Section 4 introduces the basic theories of reinforcement learning. In Section 5, we model this problem in the framework of Q-learning and propose the auto-scaling algorithm in detail. Experimental results discussed in Section 6 demonstrate the performance of our approach in different cloud market settings. Finally, we summarize our contribution and present directions for future work in Section 7.

2. Related Work

In recent years, the research on auto-scaling technologies for cloud-based applications has been studied intensively [3, 4]. Various approaches and mechanisms for different purposes have been proposed in both academia and industries, and some of them have been applied in the real market [5, 6]. Main approaches include rule-based or threshold-based strategies [7], predictive-based mechanisms [8], and heuristic methods [9]. Similar to our purpose, [10, 11] try to help SaaS providers realize optimal resource allocation to their applications. Customer requests in their model are defined as workflows or a set of independent jobs, which is different from our workload-based model. Moreover, they assume IaaS providers only offer on-demand VM instances to SaaS providers, ignoring the influence of different pricing patterns. References [12, 13] take into account this point and use game theoretic methods to generate optimal service provisioning strategies for SaaS providers. However, in their model, customer workloads have been known in advance.

Most application auto-scaling decisions happen in a dynamic, continuous, and stochastic cloud environment, which can be viewed as Markov Decision Processes (MDPs). Therefore, recent research has been focusing on applying automatic decision theoretic planning techniques such as reinforcement learning in this field. Similar to our work, [14, 15] use Q-learning approach to achieve autonomic resource allocation for auto-scaling applications. Compared with their studies, we take into account different pricing patterns of VM instances that are not considered in their work. References [16, 17] consider multiple IaaS resources pricing patterns, but they mainly study how to generate optimal bidding strategies for spot VM instances. This is quite different from our purpose: helping SaaS providers make appropriate VM renting decisions in different single pricing markets and a hybrid pricing market.

3. Problem Statement

3.1. System Model and Assumptions. As shown in Figure 2, there are three roles in a three-tire cloud market: IaaS providers, SaaS providers, and cloud application customers. SaaS providers as the middle layer in this market rent cloud resources from IaaS providers to host and run their web applications. At the same time, they provide available SaaS services to their end users. In this paper, we study a simplified model which only consists of one IaaS provider R_{iaas} and one SaaS provider R_{saas}.

Similar to Amazon EC2 service, IaaS provider R_{iaas} adopts different pricing plans to charge SaaS provider R_{saas}. Here we assume that there are two kinds of virtual machine (VM) instances: on-demand VM and reserved VM. On-demand VM instances are charged by a fixed price for each charge unit. SaaS providers could adjust the number of VM instances depending on their actual demands at any time. Reserved VM instances offer a discount compared with the price of on-demand instances. However, they are not as flexible as on-demand VM instances. SaaS providers have to rent VM instances with a long-term commitment. In addition, the price of VM instances is not only related to the pricing plans but also affected by the configurations of VM (e.g., CPU and memory).

Generally, customers' workload can be classified into two types: job-based workload and application-based workload. Job-based workload implies that customers submit a set of jobs to web applications and SaaS providers need to allocate appropriate resources to each job. Application-based workload is more general. SaaS providers make resource scaling decision based on the whole application workloads rather than specific jobs. Since job-based workload can be abstracted into application-based workload to some extent, in this paper, we model customers' workload as the second form.

3.2. Problem Formulation. From the perspective of SaaS providers, an inevitable problem that needs to be dealt with is determining the optimal renting policy to meet their end-users' demands while maximizing their utilities. In a dynamic

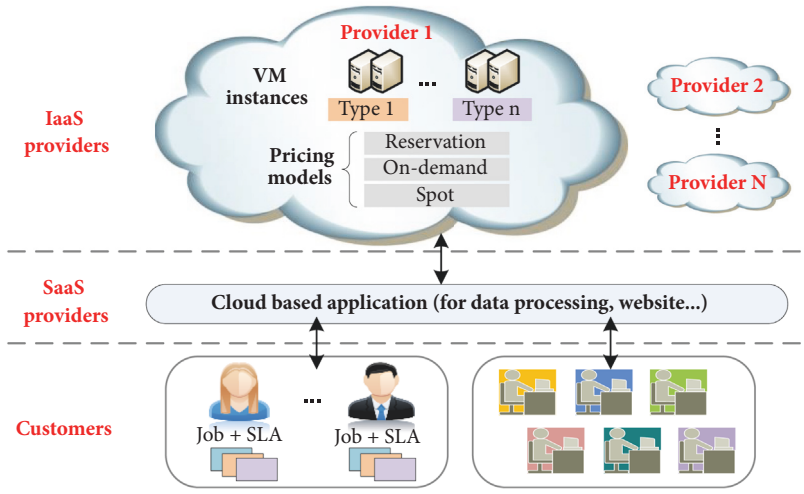


Figure 2: Three-tire cloud market model.

cloud environment, customers' workload always changes, which makes the decision more difficult. In this part, we formulate this problem in detail.

(i) IaaS Resources Model. IaaS provider R_{iaas} offers infinite VM instances in this market. These VM instances have different capacities and different pricing policies are applied to them. There are totally N types of VMs according to the VM configurations. For per unit of time T_{period}, the processing capacity of j-th type of VM instances is C_j, and the prices of on-demand instance and reserved instance are P_o^j and P_r^j, respectively. Normally, P_o^j is higher than P_r^j. Because of the difference in price, on-demand VM instances and reserved VM instances have different renting rules, which are reflected in the minimum rental period. On-demand VM instances have a shorter rental period T_{period} than the period $T_{reservation}$ for reserved VM instances. SaaS providers are not allowed to cancel their existing renting reserved VM instances when these VM instances are still alive during $T_{reservation}$.

(ii) Customers and Workloads. There are M customers in this market. Since we adopt the application based workload model, we view all customers as a whole. They request service from SaaS provider R_{saas} and pay a fixed fee P_{usage} for per T_{period}. At time t, service request from all customers is W_t. As we mentioned in Section 1, customer workloads normally have some certain regularity within a long period (e.g., Wikimedia workload showed in Figure 1). Thus here we use $T_{workload}$ to describe the workload pattern, i.e., the cyclic variations of many real-world application workloads. Customer requests in different $T_{workload}$ follow the similar pattern.

(iii) SaaS Provider Model. In order to satisfy with the customer requests, SaaS provider R_{saas} needs to rent sufficient IaaS resources from IaaS provider R_{iaas} to run a web application R_{app}. At each decision time T_i, where $i \in [1, 2, \ldots]$ and $T_{i+1} - T_i = T_{period}$, SaaS provider R_{saas} is able to adjust his renting policy depending on the change of customer workloads and his current knowledge. A renting policy involves two aspects: the choice of pricing method and the number of required different types of VM instances. We use a tuple $(X_{T_i}^{jO}, X_{T_i}^{jR})$ to represent the new generated renting policy at time T_i, where $X_{T_i}^{jO}$ is the number of rented on-demand VM instances with type j for period $[T_i, T_{i+1}]$, and $X_{T_i}^{jR}$ is the number of new rented reservation VM instances with type j for $[T_i, T_i + T_{reservation}]$.

The total processing capacity offered by application R_{app} at time t is calculated as

$$totalC_t = \sum_{j=1}^{N} \left(V_t^{jO} + V_t^{jR}\right) * C_j \quad (1)$$

where $t \in [T_i, T_{i+1}]$ and V_t^{jO} and V_t^{jR} are the number of rented on-demand VM instances and reserved VM instances with type j during $[T_i, T_{i+1}]$, respectively. According to the definition of the renting policy, $V_t^{jO} = X_{T_i}^{jO}$ and V_t^{jR} can be defined as

$$V_t^{jR} = V_{T_i}^{j\_AliveR} + X_{T_i}^{jR} \quad (2)$$

where $V_{T_i}^{j\_AliveR}$ is the number of alive reserved VM instances with type j at time T_i. Its computational formula is as follows:

$$V_{T_i}^{j\_AliveR} = V_{T_{i-1}}^{j\_AliveR} - V_{T_i}^{j\_notAliveR} \quad (3)$$

where $V_{T_{i-1}}^{j\_AliveR}$ is the number of alive reserved VM instances with type j at time T_{i-1} and $V_{T_i}^{j\_notAliveR}$ is the number of expired reserved VM instances with type j until T_i.

The goal of SaaS provider R_{saas} is trying to rent appropriate VM instances at different time to maintain the performance of application R_{app} as close as possible to the target

demand given by customers while minimizing the renting cost paid to IaaS service R_{iaas}. If the future workloads are known in advance when R_{saas} is making the renting policy decisions, this decision-making problem is easy to solve. However, customer workloads are dynamic and unforeseeable. SaaS provider normally has to decide his renting policy before the next workload is coming. All these factors make web application auto-scaling become a difficult objective for SaaS provider R_{saas} to achieve.

4. Theoretical Foundations

Reinforcement learning (RL) is an important method in the field of machine learning and intelligent control [18]. In a dynamic and unknown environment, an agent with no prior knowledge takes an action to change its state and then gets an instantaneous reward from environment which reflects the quality of this action. Based on the feedback and current state, this agent makes a decision for the next action. Through constant trial-and-error interactions with the environment, this agent is able to learn the optimal actions for different states.

In general, because of the uncertainty and randomness properties, reinforcement learning problems can be modeled as Markov Decision Processes (MDPs), which describe a fully observable environment for reinforcement learning [19]. A Markov Decision Process can be defined by a five tuple: $\mathscr{M} = \langle \mathcal{S}, \mathscr{A}, \mathscr{P}, \mathscr{R}, \gamma \rangle$. $\mathcal{S}$ is a set of environmental states. $\mathscr{A}$ is a set of actions. $\mathscr{P}_a(s, s') = \mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$ represents the probability that action a in state s at time t will move to a new state s' at time t + 1. $\mathscr{R}$ is a reward function, where $\mathscr{R}_a(s, s') = \mathbb{E}(s_{r+1} \mid s_t = s, a_t = a, s_{t+1} = s')$ is the immediate reward obtained by transferring from state s to s' with action a. $\gamma \in [0, 1]$ is a discount factor which is used to balance the influence of present reward and future rewards. In order to find the optimal policy $\pi^*(s)$ in state s, the optimal value function is defined as $V^*(s) = \max_\pi \mathbb{E}(\sum_{t=0}^{\infty} \gamma^t \mathscr{R}_t(s_t, s_{t+1}))$, where π is a complete decision policy for following steps. This function can be rewritten as $V^*(s) = \max_a(\mathscr{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s')\mathrm{V}(s'))$, $\forall s \in \mathcal{S}$. Therefore, the optimal policy for state s can be specified as $\pi^*(s) = \mathrm{argmax}_a(\mathscr{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s')\mathrm{V}(s'))$. As long as the optimal value function is found, the optimal policy can be obtained accordingly.

If the entire environment model is known, a MDP problem can be easily resolved by some dynamic programming methods, such as value iteration and policy iteration. However, in most cases, the state transition probability function and reward function are not known in advance. Q-learning [20] is a model-free reinforcement learning algorithm which can be used to find optimal policies by learning from previous decision-making experiences. It does not rely on complete a priori knowledge of the environment. Following the basic idea of reinforcement learning, agents constantly perform actions in different states and then observe state transitions and relevant rewards. Derived from the optimal value function we mentioned before, Q-learning uses Q function to approximate the real benefit of each state-action pair <a, s> (i.e., taking action a in state s) by updating Q values during learning processes. The update rule is defined as follows:

$$\begin{aligned} &\mathrm{Q}(s_t, a_t) \\ &\quad = \mathrm{Q}(s_t, a_t) \\ &\qquad + \alpha \left[r_{t+1} + \gamma \max_a \mathrm{Q}(s_{t+1}, a_{t+1}) - \mathrm{Q}(s_t, a_t) \right] \\ &\quad = (1 - \alpha)\,\mathrm{Q}(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a \mathrm{Q}(s_{t+1}, a_{t+1}) \right] \end{aligned} \tag{4}$$

where $\alpha \in [0, 1]$ is the learning rate, determining the importance of new observed information compared with what has already been learnt so far. $\gamma \in [0, 1]$ describes how much future reward affects current decision. It represents the weight of immediate gains and long-term benefits. All these up-to-date Q values as an important part of known information for agents are stored in real time.

At any decision time, there are two different strategies for an agent to generate the next action. One is greedy policy, which means that agent will choose the optimal action based on his current knowledge. Another method is to select an action randomly. Although this policy may result in a bad payoff, it is able to update the existing experience information and detect the changes of environment. In view of this, we use ε-greedy policy which combines the above two strategies together during the whole learning process.

5. Self-Adaptive Cloud Resources Renting Approach for SaaS Providers

In a dynamic cloud market, customer workload is always changing. Faced with this environment, SaaS providers have to take IaaS service purchase decisions regularly to maximize their utility. Since these decisions have some uncertainty and they are affected by each other, we model the SaaS provider resource renting problem as a MDP. And of course it can be solved by Q-learning method. In this section, we will first describe the key elements of this Q-learning problem and then present our self-adaptive cloud resources renting algorithm.

5.1. State Space and Actions. The state space S contains all possible states that the SaaS provider R_{saas} may experience. Each state $s \in S$ provides sufficient information to describe the current situation of R_{saas} in this market. We define each state as a 3-tuple: $s = (cw, vm, time)$, where

(i) cw is the average customer workload in last decision period;

(ii) vm is a set which records the number of VM instances of each type that the SaaS provider R_{saas} owns at present;

(iii) $time$ is a specific time within the workload period $T_{workload}$.

Given a state $s \in S$, the corresponding action set $A(s)$ includes all available actions that can be taken. Here each action at state s is expressed as a=($V_s^{1O}, V_s^{1R}, \ldots, V_s^{jO}, V_s^{jR}, \ldots, V_s^{NO}, V_s^{NR}$), where V_s^{jO} is the number of rented

on-demand VM instances of type j for the next period T_{period} and V_s^{jR} is the number of new rented reservation VM instances of type j.

5.2. Reward Function. For the SaaS provider R_{saas}, it is important to provide adequate processing capacity to meet the dynamic service requests which are in the form of continuous workloads from customers. If current processing capacity offered by R_{saas} is lower than the demand of customers, customer satisfaction degree of this application R_{app} will decline. On the contrary, if SaaS provider R_{saas} allocates too many VM instances to application R_{app}, the application processing capacity will become much higher than current customer workload. In this case, SaaS provider R_{saas} not only overspends on renting VM instances, but also wastes cloud resources. Both the above inappropriate renting behaviors have adverse consequences on SaaS provider R_{saas}'s utility. Only when application R_{app} possesses suitable processing capacity which is closed to the actual amount of customer workload, SaaS provider R_{saas} can obtain a great utility. Considering these factors, we propose a reward function which can describe the SaaS provider's utility properly. At a decision time T_i, given a state $s = (cw_{T_i}, vm_{T_i}, time_{T_i})$, an action a=$(V_s^{1O}, V_s^{1R}, \ldots, V_s^{jO}, V_s^{jR}, \ldots, V_s^{NO}, V_s^{NR})$ and the next state $s' = (cw_{T_i+1}, vm_{T_i+1}, time_{T_i+1})$, if current workload is w_{T_i}, the reward function $R(s', a)$ is given by the following equations:

$$R\left(s', a\right) = Profit\,(a) + Performance\,(a) \tag{5}$$

$$Profit\,(a) = P_{usage} - \left(P_{allO} + P_{allR}\right) \tag{6}$$

$$P_{allO} = \sum_{j=1}^{N} \left[P_O^j * \mathrm{X}_{\mathrm{T}_i}^{jO} + \begin{cases} 0, & \text{if } \mathrm{X}_{\mathrm{T}_i}^{jO} \le \mathrm{X}_{\mathrm{T}_{i-1}}^{jO} \\ P_{newO} * \left(\mathrm{X}_{\mathrm{T}_i}^{jO} - \mathrm{X}_{\mathrm{T}_{i-1}}^{jO}\right), & \text{else} \end{cases} \right] \tag{7}$$

$$P_{allR} = \sum_{j=1}^{N} \left[P_r^j * \left(\mathrm{V}_{T_i}^{j\text{-}AliveR} + \mathrm{X}_{T_i}^{jR}\right) + \begin{cases} 0, & \text{if } \mathrm{X}_{\mathrm{T}_i}^{jR} \le \mathrm{V}_{T_i}^{j\text{-}notAliveR} \\ P_{newR} * \left(\mathrm{X}_{\mathrm{T}_i}^{jR} - \mathrm{V}_{T_i}^{j\text{-}notAliveR}\right), & \text{else} \end{cases} \right] \tag{8}$$

$$Performance\,(a) = \begin{cases} \mathrm{P}_{\mathrm{bonus}}, & if\ 0 \le idleVM_{T_i} \le 1 \\ \mathrm{P}_{\mathrm{bonus}} * \left(1 - \dfrac{\mathrm{idleC}_{T_i}}{C_{T_i}}\right), & if\ idleVM_{T_i} > 1 \\ -\mathrm{P}_{penalty} * \left(1 + \dfrac{-\mathrm{idleC}_{T_i}}{w_{T_i}}\right), & else \end{cases} \tag{9}$$

$$\mathrm{idleC}_{T_i} = totalC_{T_i} - w_{T_i} \tag{10}$$

$$avgC_{T_i} = \frac{\sum_{j=1}^{N} C_j * \left(\mathrm{X}_{\mathrm{T}_i}^{jO} + \mathrm{X}_{\mathrm{T}_i}^{jR}\right)}{\sum_{j=1}^{N} \left(\mathrm{X}_{\mathrm{T}_i}^{jO} + \mathrm{X}_{\mathrm{T}_i}^{jR}\right)} \tag{11}$$

$$idleVM_{T_i} = \frac{\mathrm{idleC}_{T_i}}{avgC_{T_i}} \tag{12}$$

$Profit(a)$ is the profit that SaaS provider R_{saas} earned by providing SaaS service to his end users. It equals the difference between his income P_{usage} from customers and the total expenses (P_{allO} + P_{allR}) that he spends on renting IaaS facilities from IaaS provider R_{iaas}. P_{allO} and P_{allR} are the costs of on-demand VM instances and reserved VM instances, respectively. Each of them involves two parts: one is the real cost and the other one is the initial cost of newly added VM instances. In reality, if a SaaS provider wants to rent a new VM instance, in addition to the VM usage charge that he has to pay to the IaaS provider, a small extra cost P_{newO} or P_{newR} regarding application deployment and VM warm-up will be incurred. Definitions of P_o, P_r, $V_{\mathrm{T_i}}^{\mathrm{j\text{-}AliveR}}$ and $V_{\mathrm{T_i}}^{\mathrm{j\text{-}notAliveR}}$ have been given in Section 3.

$Performance(a)$ is the gain of application performance, which depends on the resource utilization. If SaaS provider R_{saas} owns sufficient VM instances to execute customer workloads, a positive reward will be received. In contrast, a penalty will be caused if application processing capacity is lower than the customers' demand. The value of reward or penalty is related to the distance between the offered processing capacity and the real customer workload. Definitions of $totalC_{T_i}$ and C_j can be found in Section 3.

5.3. Action Selection and State Transition. Since we use the above reward function to calculate reward $R(s', a)$ and then use it to update the corresponding Q value, the newest Q value for each state-action pair represents an estimation of the future utility that the SaaS provider will achieve. As we explained in Section 4, we adopt the ε-greedy policy. Because SaaS provider wants to maximize his utility, the optimal action based on current knowledge $a^* = \arg\max_a Q(s_{T_i}, \mathrm{a})$ will be chosen with a high probability 1−ε. Besides, in order to explore more possible state-action choices, a stochastic action instead of the best one will be taken with a low probability ε.

Once SaaS performs a specific action a_{T_i}, the state will change accordingly, from s_{T_i} to $s_{T_{i+1}}$. Meanwhile, the Q value of state-action pair $< s_{T_i}, a_{T_i} >$ will be updated based on current reward value and next state $s_{T_{i+1}}$. The update formula has been given in (4).

5.4. Self-Adaptive Cloud Resources Renting Approach. In order to help SaaS provider R_{saas} generate an appropriate renting plan in a dynamic cloud market, a Q-learning based self-adaptive renting plan generation algorithm is proposed (Algorithm 1). With the idea of Q-learning, SaaS provider R_{saas} keeps learning from previous renting experiences and enriching its knowledge. This accumulated information can help R_{saas} know the best choices in different situations and then generate an efficient renting policy for each decision period.

6. Experimental Design and Analysis

In this section, we will evaluate our Q-learning based self-adaptive renting plan generation algorithm through a series

Input: Initialize parameters ε, α *and* γ
StateNum⟵1, s⟵ an initial state, *S[StateNum]*⟵s, add *A(s)* to A
For $\forall a \in A(s)$, initialize Q-values $Q(s, a)$
Output: action a for each renting decision period
(1) **loop** (for each renting decision period)
(2) // choose an action from *A(s)* using ε-Greedy policy
(3) **if** Random(0,1) < ε **then** // exploration
(4) a⟵*random A(s)*
(5) **else** //exploitation
(6) a⟵$\arg\max_a Q(s, a)$
(7) **end if**
(8) submit renting plan(i.e. action a) to IaaS provider R_{iaas}
(9) observe customer workload w over this period
(10) move to new state s'
(11) **if** $s' \notin S$ **then** // add new state into state space
(12) *StateNum*++
(13) *S[StateNum]*⟵ s'
(14) For $\forall a' \in A(s')$, initialize Q-values $Q(s', a')$
(15) **end if**
(16) calculate reward r⟵ $R(s', \text{a})$ using (5) - (12)
(17) update $Q(s, a) \longleftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
(18) $s \longleftarrow s'$
(19) **end loop**

Algorithm 1: Q-learning based self-adaptive VM instances renting algorithm.

of experiments. First, we describe the basic simulated cloud environment and present two other IaaS facilities renting strategies which will be used as baselines to evaluate our proposed algorithm in the following experiments. Then we will mainly study the performance of Q-learning based self-adaptive renting plan generation algorithm under different cloud market settings. We implement all algorithms in Matlab R2014b. All experiments were carried out on a MacBook Pro with 2.7 GHz Intel Core i5 CPU and 8GB RAM.

6.1. Experimental Setup. In our experiment, we simulate a dynamic cloud environment which involves one IaaS provider R_{iaas}, one SaaS provider R_{saas}, and some SaaS service customers. Based on the model we formulated before, default experimental parameters are set as follows.

Attributes of Cloud Resources. IaaS provider R_{iaas} has 2 types of VMs and employs two pricing methods. The rental costs ($ per hour) of on-demand VM instances and reserved VM instances are $P_o^1 = 0.094$, $P_o^2 = 0.2$ and $P_r^1 = 0.072$, $P_r^2 = 0.13$, respectively. In terms of the VM rental period, we set $T_{period} = 1h$ and $T_{reservation} = 3h$. $C_1 = 200$ and $C_2 = 400$ are the processing capacities of VM instances of type 1 and VM instances of type 2, respectively.

Customer Workload Settings. Similar to the workload model used in [14], we assume customer workloads are stochastic but follow a certain regularity in period $T_{workload}$. Here we set $T_{workload} = 24h$. The workload in each period $T_{workload}$ is a series of discrete data which obeys a normal distribution with some noise, varying in [3000, 8000]. The noise is generated randomly in interval [-300, 300]. Customers submit requests hourly so there are totally 24 workloads for 0 to 23 o'clock in a day. SaaS provider R_{saas} adjusts the renting policy every hour before the customer workload coming. That is to say, R_{saas} does not know customers' future workloads in advance.

Reward Function Parameters. Reward function represents the utility of SaaS provider R_{saas}. As for the profit part, we set the newly added costs of VM instances as fractions of the actual rental costs: $P_{newO} = P_o/60$, $P_{newR} = P_r/60$. P_{usage} is the revenue that SaaS provider R_{saas} obtains from end users. In this model, we assume P_{usage} is a constant. For example, R_{saas} has fixed number of customers and they are charged annually. Therefore, we can ignore P_{usage} from the reward function and set its value to zero. In terms of the parameters of performance, the values of positive reward and penalty with regard to resource utilization are given by $\mathrm{P}_{\mathrm{bonus}} = 2$ and $\mathrm{P}_{penalty} = 1$, respectively.

Q-Learning Parameters. In our proposed scenario, we set the values of learning rate α and discount factor γ to be both 0.5. The value of ε is 0.1 for ε-greedy policy. According to the value range of customer workloads, we set the maximum number of rented VM instances for each type to be 28.

Since customer workloads change regularly in 24 hours (i.e., $T_{workload}$), we use everyday reward of SaaS provider R_{saas} as the evaluation indicator for these experiments. Because of the randomness of customer workloads, each experiment was repeated 50 times and the average results will be presented.

Baseline Algorithms. In order to measure the performance of our algorithm (QAA), we will compare it with another two algorithms with different renting strategies: empirically

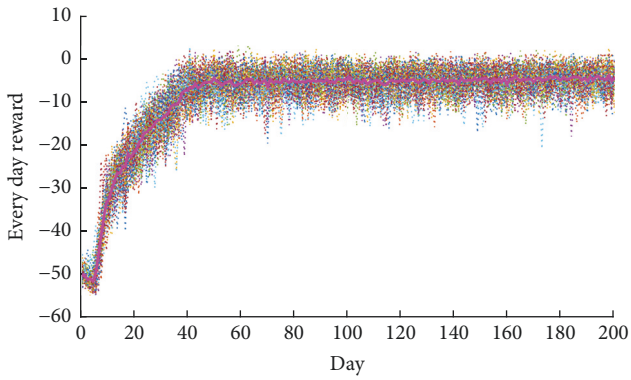


FIGURE 3: Everyday reward of R_{saas} in on-demand market.

based adjustment algorithm (EAA) and threshold-based adjustment algorithm (TAA).

Empirically based adjustment algorithm adopts a simple and direct strategy to generate new renting policy. Because SaaS provider R_{saas} does not know the upcoming customer workload when making decisions, he adjusts the number of rental VM instances according to the last workload. Specifically, at decision time T_i, SaaS provider R_{saas} will rent $w_{T_{i-1}}/avgC_{T_{i-1}}$ VM instances from IaaS provider R_{iaas}.

Threshold-based adjustment algorithm is similar to empirically based adjustment algorithm. It also refers to the last customer workload but does not change the renting policy each time. Only when the difference between customer workload and processing capacity offered by R_{saas} exceeds a specific threshold, a new renting policy will be generated. The basic rules are given as follows:

(i) If $|totalC_{T_{i-1}} - w_{T_{i-1}}| > w_{threshold}$ and $totalC_{T_{i-1}} - w_{T_{i-1}} < 0$, then rent more VM instances to fit $w_{T_{i-1}}$.

(ii) If $|totalC_{T_{i-1}} - w_{T_{i-1}}| > w_{threshold}$ and $totalC_{T_{i-1}} - w_{T_{i-1}} > 0$, then rent less VM instances to fit $w_{T_{i-1}}$.

(iii) If $0 \le |totalC_{T_{i-1}} - w_{T_{i-1}}| \le w_{threshold}$, then maintain previous renting policy.

We set $w_{threshold} = thresholdRate * avgC_{T_{i-1}}$, where $thresholdRate = 0.2$.

6.2. Experiments of On-Demand VMs Market. In this part, we consider a simple cloud market environment where IaaS provider R_{iaas} only provides on-demand VM instances to SaaS provider R_{saas}. For each decision period, SaaS provider R_{saas} uses Q-learning based self-adaptive renting plan generation algorithm to readjust his renting policy to adopt the workload change.

Figure 3 shows the variation of SaaS provider's everyday reward in 200 days. Each colored dash line is the actual everyday reward curve for one test and the solid magenta line is the average results of 50 experiments. It is clear that SaaS provider's everyday reward is lower at the early stage of decision-making periods but it keeps rising and then rapidly reaches a high and relatively stable level.

In order to observe more clearly, we show some detailed information in Figure 4. Figures 4(a)–4(d) depict the comparison results of customer workload and SaaS provider's offered capacity in days 20, 40, and 60 and the last day, respectively. In day 20, SaaS provider R_{saas} prefers to rent a small number of VM instances even though his capacity does not meet customers' demand. This results in a low reward in return in most of the day. As shown in Figures 4(b)–4(d), we can find that the gap between customer workload and R_{saas} owned capacity becomes smaller and the capacity is always higher than customer workload as time goes by. The reason for this series of changes is that with the help of Q-learning based self-adaptive renting plan generation algorithm, the SaaS provider gets more trading experiences. This accumulated knowledge suggests him to generate optimal renting policy in most cases.

For the purpose of estimating the performance of our proposed algorithm, we compare it with the other two mentioned algorithms EAA and TAA. As maximizing SaaS provider's utility is the optimization target of this model, we focus on comparing three algorithms with regard to everyday reward of R_{saas}. Figure 5 presents the average everyday reward curves for three renting decision-making processes using different algorithms. Compared with algorithms EAA and TAA, our proposed algorithm does not perform well during the initial learning phases. However, after a period of time (around day 47), a higher reward could be obtained and this leading position still be kept in the following days.

As we mentioned before, because of the nature of Q-learning, our approach could adapt to the market changes automatically. In this experiment, we use two periodic stochastic workloads to estimate its performance in this respect. In the first phase (for day 1 to day 1250), we still use the default customer workloads which vary in [3000, 8000]. From day 1251, the workloads are changed to another value range [2500,6000] with the same noise generation method that we apply in default workloads. Figure 6 displays the changes of SaaS provider's everyday reward throughout the whole period, which also reflects the adaptive ability of our algorithm. In both phases, reward values first increase gradually and then reach a stable level.

6.3. Experiments of Reserved VMs Market. In this part, we apply Q-learning based self-adaptive renting plan generation algorithm into another simulated cloud market. In this scenario, SaaS provider R_{saas} only rents reserved VM instances from IaaS provider R_{iaas}.

Compared with the flexibility of on-demand VM instances, reserved VM instances have a restriction on the minimum rental period. It is obvious that the longer this period (i.e., $T_{reservation}$) is set, the less flexibility is given to the SaaS provider. This is because SaaS provider R_{saas} is not allowed to reduce the number of VM instances that are still within the validity period, even if he has no further use for so many resources. On this account, we set different values to $T_{reservation}$ to study the relationship between everyday reward and the length of period $T_{reservation}$. Figure 7(a) depicts reward curves of R_{saas} under different settings of $T_{reservation}$. We can find that no matter the length of

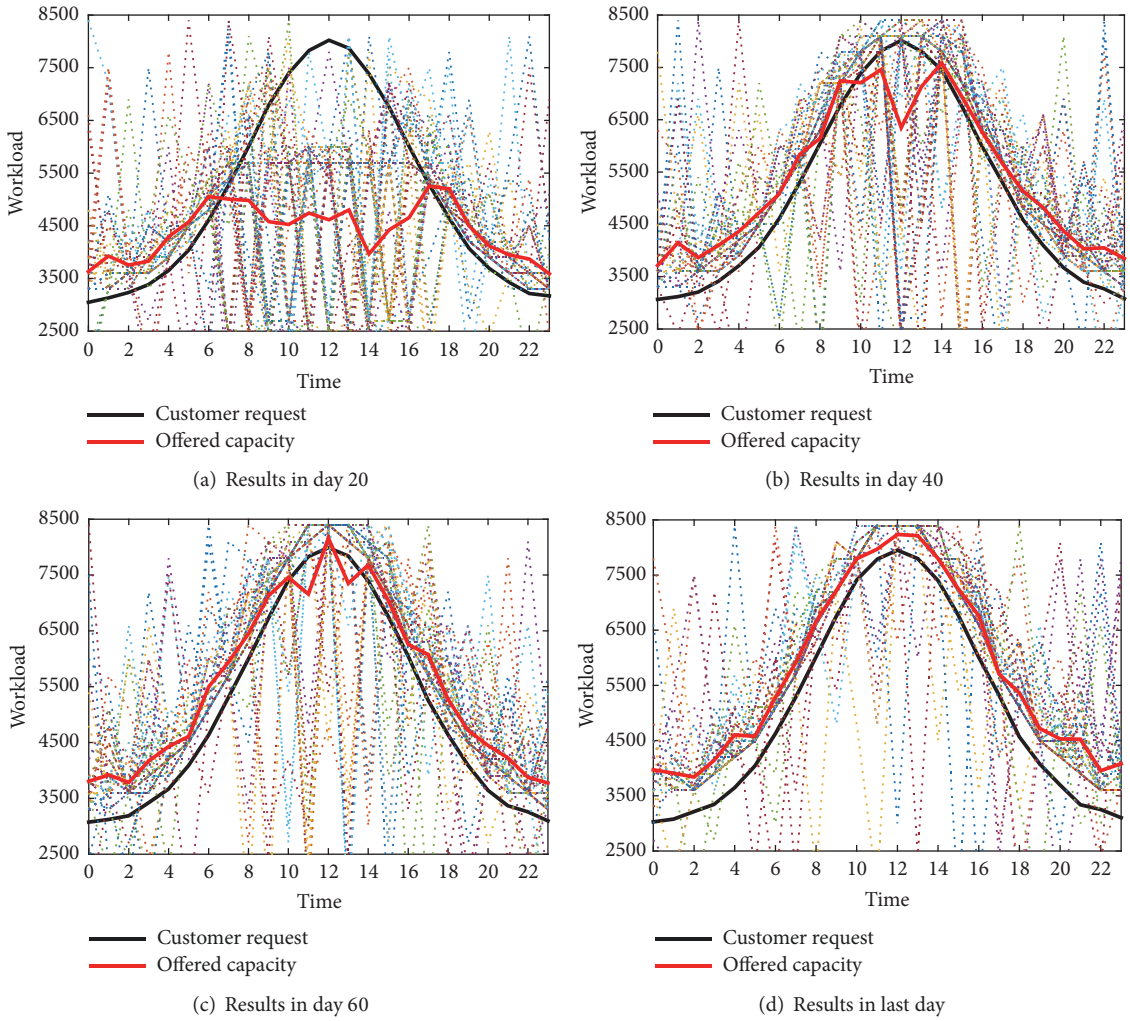


Figure 4: Comparisons between customer workload and offered capacity in different days.

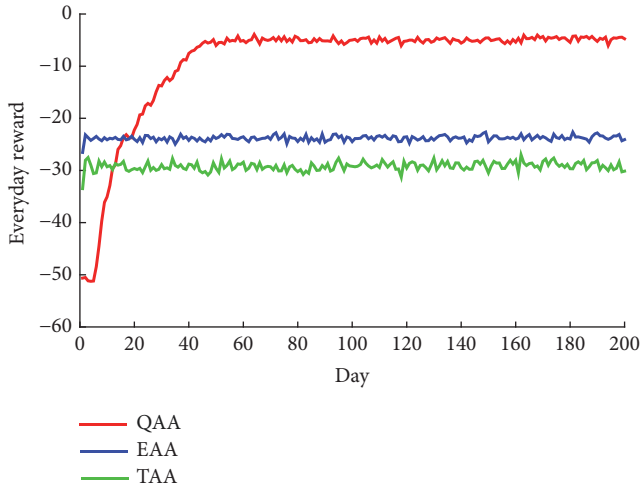


Figure 5: Performance comparison in on-demand VMs market.

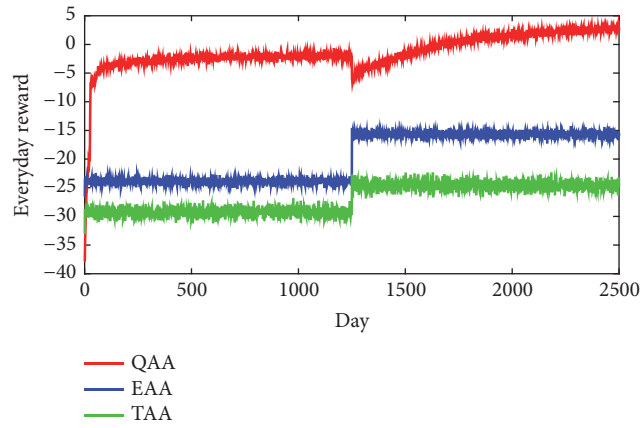
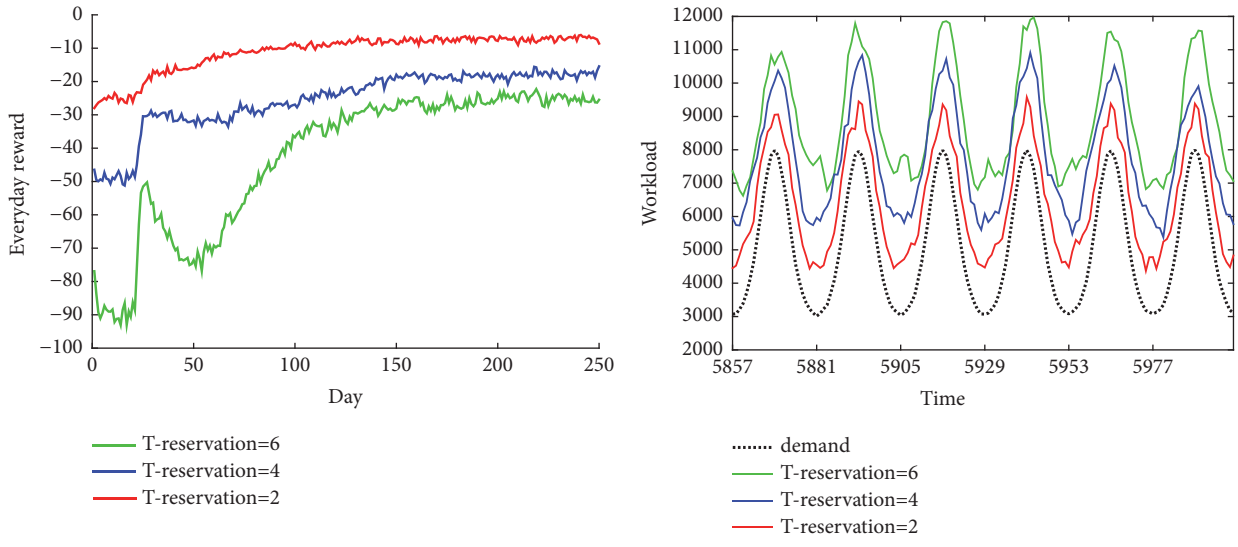


FIGURE 6: Reward variation under different customer demands.



(a) Reward with different $T_{reservation}$

(b) Workload with different $T_{reservation}$

FIGURE 7: Comparison results regarding different $T_{reservation}$.

$T_{reservation}$, each reward curves can reach a stable level in the end. It means that our proposed algorithm is still effective on generating appropriate policy for R_{saas} in reserved VMs market. Moreover, it is clear that, for a specific day, a higher reward can be obtained with a shorter $T_{reservation}$. This is because, with a shorter minimum rental period, R_{saas} has more flexibility to adjust the number of owned VM instances to respond to the changes of customer demand at different times. Figure 7(b) compares the customer workload curve with the offered processing capacity curves corresponding to different $T_{reservation}$ in the last 6 days. As shown in this graph, the capacity curve with a shorter $T_{reservation}$ is closer to the real customer workload. Accordingly, its corresponding reward curve in Figure 7(a) is higher than others.

Figure 8 presents the performance comparison results of three different algorithms regarding achieved everyday reward. In this experiment, we set $T_{reservation} = 3$ which is the default value we defined in Section 6.1. Similar to the observations that we get in Figure 5 (i.e., comparison results in on-demand VMs market), our algorithm can generate better renting policy in the long run compared with algorithms EAA and TAA.

6.4. Experiments of Hybrid Market. In previous experiments, we evaluate Q-learning based self-adaptive renting plan generation algorithm in on-demand VMs market and reserved VMs market separately. In this part, we consider a more complex environment that IaaS provider R_{iaas} adopts two pricing models in a hybrid market. Compared with on-demand VMs market, SaaS provider R_{saas} is able to acquire reserved VM instances to reduce his renting cost. Meanwhile, he can adjust his renting policy more freely than in a market where IaaS provider R_{iaas} only provides reserved VM instances. Figure 9 displays the performance of SaaS provider's reward in the on-demand VMs market, reserved VMs market, and hybrid market. We can find that, faced with the same customer workloads, R_{saas} could gain more rewards

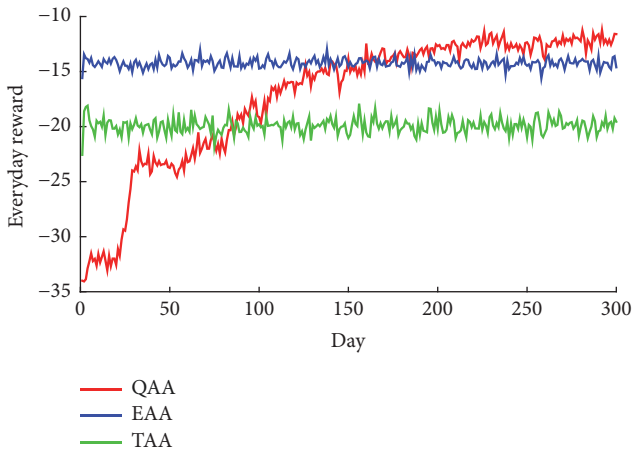


Figure 8: Performance comparison in reserved VMs market.

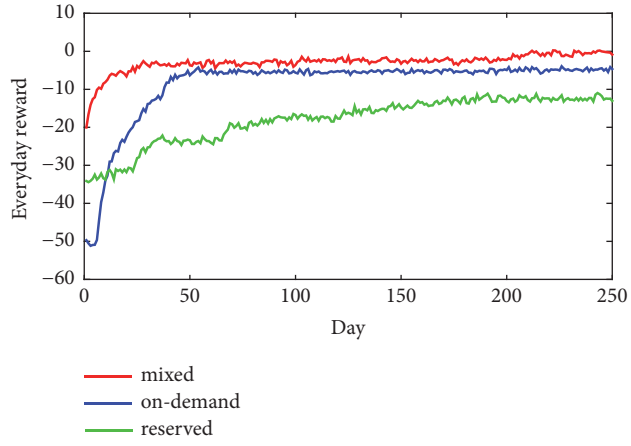


Figure 9: Performance comparison in different market settings.

if he rents both on-demand VM instances and reserved VM instances from R_{iaas}.

7. Conclusion and Future Work

In this paper, we proposed a cloud application auto-scaling approach based on Q-learning method to help SaaS providers make optimal resource allocation decisions in a dynamic and stochastic cloud environment. Unlike existing studies, we took into account different VM pricing mechanisms in our model, including on-demand pattern and reserved pattern. Through a series of experiments, we demonstrated the effectiveness of our algorithm and evaluated its performance in different market settings. Nevertheless, auction-based pricing mechanism plays an important role in the cloud trading market. In the future, we will add spot VMs into our model and design appropriate algorithms for a more complex heterogeneous cloud market.

Data Availability

All the underlying data related to this article are available upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The authors would like to acknowledge the support provided by the National Natural Science Foundation of China (61872222), National Key Research and Development Program of China (2017YFA0700601), the Key Research and Development Program of Shandong Province (2017CXGC0605, 2017CXGC0604, and 2018GGX101019), and the Young Scholars Program of Shandong University.

References

[1] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad, "Cloud computing pricing models: a survey," *International Journal of Grid and Distributed Computing*, vol. 6, no. 5, pp. 93–106, 2013.

[2] "Page view statistics for wikimedia projects," 2015, http://dumps.wikimedia.org/other/pagecounts-raw/.

[3] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[4] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: a taxonomy and survey," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–33, 2018.

[5] "Amazon auto scaling," http://aws.amazon.com/autoscaling/.

[6] "Rightscale," http://www.rightscale.com/.

[7] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '12)*, pp. 644–651, May 2012.

[8] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.

[9] T. Chen and R. Bahsoon, "Self-adaptive trade-off decision making for autoscaling cloud-based services," *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 618–632, 2017.

[10] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS '13)*, pp. 67–78, May 2013.

[11] Z. Liu, S. Wang, Q. Sun, H. Zou, and F. Yang, "Cost-aware cloud service request scheduling for SaaS providers," *The Computer Journal*, vol. 57, no. 2, pp. 291–301, 2014.

[12] V. D. Valerio, V. Cardellini, and F. L. Presti, "Optimal pricing and service provisioning strategies in cloud systems: A stackelberg game approach," in *Proceedings of the IEEE 6th International Conference on Cloud Computing (CLOUD '13)*, pp. 115–122, July 2013.

[13] D. Ardagna, B. Panicucci, and M. Passacantando, "A game theoretic formulation of the service provisioning problem in cloud systems," in *Proceedings of the 20th International Conference on World Wide Web (WWW '11)*, pp. 177–186, ACM, April 2011.

[14] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic

resource allocation in clouds: towards a fully automated workflow," in *Proceedings of the Seventh International Conference on Autonomic and Autonomous Systems (ICAS '11)*, pp. 67–74, 2011.

[15] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[16] N. Borissov, D. Neumann, and C. Weinhardt, "Automated bidding in computational markets: An application in market-based allocation of computing services," *Autonomous Agents and Multi-Agent Systems*, vol. 21, no. 2, pp. 115–142, 2010.

[17] M. Abundo, V. Di Valerio La Sapienza, V. Cardellini, and F. L. Presti, "QoS-aware bidding strategies for VM spot instances: A reinforcement learning approach applied to periodic long running jobs," in *Proceedings of the 14th IFIP/IEEE International Symposium on Integrated Network Management (IM '15)*, pp. 53–61, May 2015.

[18] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[19] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, 2014.

[20] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.