Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A Q-learning based auto-scaling approach for provisioning big data analysis services in cloud environments

Shihao Song, Li Pan*, Shijun Liu*

School of Software, Shandong University, Jinan, China

ARTICLE INFO

Keywords:
Auto-scaling
Big data analysis services
Reinforcement learning
Cloud computing
Cost optimization
Dynamic workloads

ABSTRACT

Currently, analyzing big data to capture hidden values in various fields is one of the most popular research directions. Analytics-as-a-Service (AaaS) providers typically construct a common platform by renting virtual machine (VM) resources from Infrastructure-as-a-Service (IaaS) providers instead of owning their own physical resources, to provision big data analysis services to end users. However, due to the fact that service demands from AaaS users tend to fluctuate over time in reality, how to dynamically adjust the type and quantity of VMs rented from IaaS providers has become an urgent problem to be solved. In this article, we assume that IaaS providers can offer VM resources in two charging modes, and service demand workloads arrive stochastically. We formulate the problem as a Markov decision process and propose a Q-Learning based auto-scaling method, which explores the trade-off between AaaS providers' cost and VM resource utilization. Eventually, we evaluate our method on both real traces and simulated traces and compare it with some existing methods. Experimental results demonstrate that our method can not only achieve the goal of reducing cost, but also ensure the high utilization of VM resources in the long run.

1. Introduction

Many data owners aim to discover the important knowledge hidden in big data and make effective decisions through analyzing these data [1,2]. However, it is extremely expensive to build the facilities needed for big data analysis by themselves. Most data owners prefer to the services provisioned by professional data analysis companies. Currently, there are a large number of big data analysis services delivered to users over the Internet. Fig. 1 illustrates the three-tier structure for such big data analysis service provisioning scenario. From the bottom to the top, there are three roles involved, which are Infrastructure-as-a-Service (IaaS) providers, Analytics-as-a-Service (AaaS) providers, and analysis service users. IaaS providers are companies like Amazon and Microsoft, which own a large number of physical machines. They provide infrastructure services in the form of $Virtual\ Machine\ (VM)\ instances$ to all individuals and groups through the network [3]. AaaS providers occupy the middle layer of the three-tier structure, and they usually do not establish their own physical clusters or infrastructure platforms, but to provision analysis services to users by renting commercial cloud infrastructure. Users of these analysis services can be individuals or companies, who submit analysis service requests to an AaaS platform and get corresponding analysis results. One of the most important features of cloud computing is its flexibility in using cloud infrastructure. Through the use of cloud resources, AaaS providers have the ability to dynamically handle big data analysis service requests that change over time. In other words, AaaS providers can flexibly scale the resources they rent in response to actual conditions to address dynamic service requests.

In order to meet analysis service requests that change over time, AaaS providers need to develop a strategy to dynamically adjust the resources they rent [4]. If the service capability provisioned by AaaS providers falls short of user demands, it will negatively impact users' experience of the service and lead to a loss of users in the long run. If AaaS providers lease a substantial number of cloud resources, its ability to serve users will be enhanced, but the cost of renting resources also increases at the same time. As a result, the service providers need to reduce the number of cloud instances they rent while ensuring the quality of service (QoS) delivered to their users. Simultaneously, there are many types of VM instances in the cloud market, which are varied in processing capacities and pricing patterns [5]. The primary cloud instance pricing patterns include on-demand, reserved, and spot, and each has its own advantages. The on-demand pattern is typically used based on actual demand, with the highest price among the three. The reserved pattern requires a partial deposit in advance and offers a discount each time when using cloud resources in this pattern, but it lacks flexibility. The spot pattern is prone to service interruptions. If more

* Corresponding authors.
E-mail addresses: panli@sdu.edu.cn (L. Pan), lsj@sdu.edu.cn (S. Liu).

https://doi.org/10.1016/j.future.2024.01.003
Received 25 March 2023; Received in revised form 1 January 2024; Accepted 2 January 2024
Available online 6 January 2024

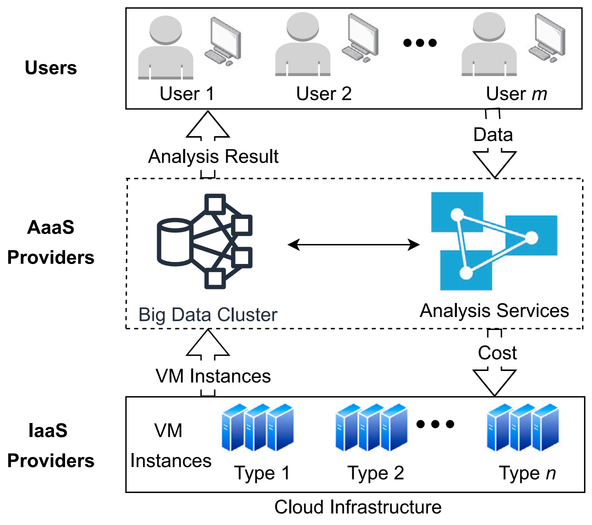


Fig. 1. Three-tier structure for big data analysis service provisioning scenario.

resources in reserved pattern are used in scenarios that require high flexibility, it will significantly degrade the user experience. Conversely, if resources in on-demand pattern are heavily used in scenarios with stable workloads, it will greatly increase the cost of resource renting.

Base on the above situations, this paper addresses a practical problem: how can AaaS providers fully utilize the advantages of on-demand and reserved charging modes to achieve cost optimization and resource utilization improvement in the face of dynamically changing analysis service requests without future request changes? Auto-scaling is designed to automatically adjust the number of resources to meet dynamic workloads, which can assist AaaS providers in making effective scaling decisions to balance cost and VM resource utilization [6]. However, some existing research on auto-scaling strategies is either based on dynamic programming mechanisms to derive scaling strategies [7,8], or based on simple resource utilization thresholds to determine scaling strategies [9]. These methods cannot effectively balance cost optimization, resource utilization improvement, and elastic scaling under multiple resource charging modes. In this work, in order to address the problem mentioned earlier, we adopt a more reasonable modeling and research approach, and make the following significant contributions in big data analysis service scenario:

- We formulate the scaling decision making process in big data analysis scenario in cloud environments as a Markov Decision Process (MDP). At the same time, we present a cost model based on on-demand and reserved charging modes.
- We propose an auto-scaling algorithm based on Q-Learning to address the problem of cloud resource scaling without knowledge of future workloads. In this algorithm, we design a unique reward function that integrates various charging modes, resource cost and resource utilization. The reward function takes into account the cost of AaaS providers and the usage of VM resources, and includes penalty for violating Service Level Agreement (SLA) as a part of cost.
- We conduct simulation experiments on real and simulated traces and compare our proposed Q-Learning based auto-scaling method with some existing baseline methods. Through a performance evaluation in terms of cost savings and resource utilization, we observe that our method can achieve a remarkable 12% reduction in cost and a significant improvement in resource utilization.

The rest of this article is organized as follows: Section 2 summarizes related work. In Section 3, we formulate the auto-scaling problem in big data analysis scenario. The proposed algorithm is shown in detail in Section 4. Meanwhile, in Section 5, we verify our algorithm by a series of experiments. Finally, Section 6 summarizes this paper and prospects our future work.

2. Related work

Big data analysis as one of the hottest industries at present. Through analyzing data, individuals and enterprises can make real-time and accurate decisions based on the valuable information obtained from the data [10]. In cloud environments, AaaS providers usually rent appropriate cloud resources from IaaS providers to provision big data analysis services required by users in different fields at a low cost, while ensuring the maximum utilization of rented resources. In a dynamic environment, it is important for AaaS providers to adjust the type and number of cloud VM instances in real time according to workloads. This problem is often called auto-scaling.

Recently, a lot of works have focused on traditional auto-scaling problems, and the primary goal is to minimize the usage cost of cloud resources. Mao et al. [11] propose two methods to solve auto-scaling problem on the premise of fixed cost budget. Their goal is to achieve optimal performance and complete users' tasks within the budget as soon as possible. Jannapureddy et al. [12] consider the dynamic workload, and propose a framework based on thresholds to dynamically adjust virtual nodes by monitoring real-time workload to obtain higher cost-effectiveness. Wang et al. [8] concentrate on the cost of renting cloud resources and propose an online algorithm with a competitive ratio of 2-α. They realize the adaptation to dynamic workload and minimize the cost by combining on demand and reserved cloud resource pricing patterns. Mao et al. [13] investigate the impact of the job completion deadline on the auto-scaling problem. By allocating resources and scheduling tasks on the most cost-effective instances, they minimize the cost of using cloud resources and ensure that all tasks could be completed before specified deadline. However, the above algorithms are carried out under the constraints of cost limitations, and the final optimization goal is relatively simple. In fact, the combination of multiple objectives for resource cost optimization will be more comprehensive and practical.

In cloud markets, there are usually three pricing patterns, on-demand, reserved and spot. Nevertheless, simultaneously considering the scaling of cloud resources of the three patterns significantly increases the complexity of solving the auto-scaling problems. Si et al. [14] study the renting process of cloud resources and propose an online auto-scaling algorithm with a competitive ratio of no more than 2. They thoroughly account for the flexible characteristics of on-demand cloud VM instances, address the problem of renting heterogeneous cloud VM instance, and ensure that the cost of renting resources is minimized. Garí et al. [15] propose an algorithm based on reinforcement learning to adjust rented resources under dynamic workload. In their paper, they consider both on-demand and spot cloud VM instances, and minimize the maximum completion time by adjusting rented cloud resources in the algorithm. George et al. [16] consider the dynamic workload in web application and predict the workload changes through the analysis of historical data. They solve the auto-scaling problem on on-demand cloud resources through the prediction mechanism, ensure the quality of service and reduce the cost of renting cloud resources. Nevertheless, all the above works only focus on one of the three cloud resource pricing patterns or a combination of on-demand and spot pricing patterns, while in this work we consider a more prevalent combination of on-demand and reserved pricing patterns.

At the same time, there are many works addressing the auto-scaling problems of cloud resources from the perspective of service providers or application providers. Horovitz et al. [17] propose an auto-scaling algorithm based on reinforcement learning to automatically adjust a threshold. They use thresholds to determine to increase or decrease cloud VM instances in their work and achieve the minimum amount of rented resources. Besides, their method finally ensures SLA and reduces the cost of Software-as-a-Service (SaaS) providers. Wei et al. [18] study web applications from the perspective of SaaS providers and fully consider different cloud VM instances and charging modes. They propose a Q-Learning based auto-scaling method to minimize the cost of using cloud VM instances and ensure the service capacity provided by SaaS providers. In order to guarantee the SLA of stream event processing and provide better scalability for cloud resources, Runsewe et al. [19] propose a hierarchical multi-dimension hidden Markov model from the perspective of big data stream application providers. However, most of the above works focus on the auto-scaling problems faced by general SaaS providers, and rarely consider the problems from the perspective of big data analysis service providers.

Although there are many ways to solve the problem of auto-scaling in web application scenarios or big data application scenarios, there are few methods dealing with the same problem in big data analysis scenario. Additionally, many existing works only consider one charging pattern or a combination of on-demand and spot patterns in the cloud markets. In our work, we focus on big data analysis service, consider two charging patterns of on-demand and reserved, and take into account cost and VM resource utilization. Our ultimate goal is to achieve a balance between cost and VM resource utilization.

3. Problem formulation

For AaaS providers, it is necessary to make a series of effective resource scaling decisions to deal with the uncertain workload in cloud environments, with the aim of minimizing cloud VM instance usage cost and maximizing the utilization of their rented instances. Improving VM resource utilization can help in reducing the amount of resources that AaaS providers need to rent, and ultimately saving cost.

At present, major cloud providers in the cloud market generally offer three types of VM instances: reserved, on-demand and spot VM instances [20]. Under the same configuration, reserved instances have the lowest unit price, but their reservation cycle is too long, which usually spans one month or one year; on-demand instances offer the highest flexibility, but their unit price is relatively high; spot instances have a moderate price and their flexibility is similar to on-demand instances, but they may have a risk of service interruptions. In our work, our goal is to make a reasonable scaling strategy, by considering both the cost and the flexibility of the instances used, while avoiding instance interruptions. Consequently, we focus exclusively on reserved and on-demand instances in this work.

We assume that IaaS providers offer n types of VM instances, which differ greatly in configurations of Central Processing Unit (CPU) and memory. We use CRs to denote the number of various types of cloud resources that AaaS providers purchase from IaaS providers, with $CRs = \{CR^o_1, \cdots, CR^o_i, \cdots, CR^o_n, CR^r_1, \cdots, CR^r_j, \cdots, CR^r_n\}$, where $i, j \in [1, 2, \ldots, n]$. CR^o_i indicates the number of the ith on-demand instance, and CR^r_j indicates the number of the jth reserved instance. The unit price of the ith on-demand instance is denoted by P^o_i, and the unit price of the jth reserved instance is denoted by P^r_j, where $i, j \in [1, 2, \ldots, n]$. Generally, P^r_i is lower than P^o_i.

Let T_k denote a decision-making moment, where $k \in [1, 2, \ldots]$. T_d represents the time between every two decision moments, and $T_d = T_{k+1} - T_k$. In this work, we have decision period T_d being equal to the minimum period of an on-demand instance. Let C_t represent total cost generated in a decision cycle, which is caused by AaaS providers to provision analysis services to users, and it can be calculated as:

$$C_t = C_v + C_s. \tag{1}$$

C_v is the cost of using all VM instances, and it depends on type, number and charging mode of instances used. For each VM instance, since the startup time is mostly in seconds, which is much less than T_d, we do not consider the impact of the instance startup time. So C_v is calculated as:

$$C_v = \sum_{i=0}^{n} P^o_i \cdot CR^o_i \cdot T_d + \sum_{j=0}^{n} P^r_j \cdot CR^r_j \cdot T_d. \tag{2}$$

SLA is an agreement on service quality between service providers and users, which is mainly an agreement on the waiting time in our work. Let $waitTime_{SLA}$ denote the maximum waiting time specified in SLA. When the services provisioned by AaaS providers cannot meet users' needs (i.e., users' waiting time for services exceeds $waitTime_{SLA}$), a penalty cost C_s will be generated. In big data analysis scenario, workload usually refers to user service requests, which usually includes dependent tasks and independent tasks. In this work, in order to fully reflect the concurrency of different VM instances, we choose independent tasks as our research object, and assume that these tasks only differ in their duration, which can be roughly divided into long tasks and short tasks. We assume that there are m tasks completed within T_d time, and let $L(l)$ represent whether the lth task violates SLA. $waitTime_l$ indicates the waiting time before the lth task begins to be executed since its submission, where $l \in [1, 2, \ldots, m]$. $L(l)$ can be calculated as:

$$L(l) = \begin{cases} 1, & waitTime_l > waitTime_{SLA} \\ 0, & waitTime_l \le waitTime_{SLA} \end{cases}. \tag{3}$$

Thus, C_s is calculated as follows:

$$C_s = \sum_{i=0}^{m} L(l) \cdot (waitTime_l - waitTime_{SLA}) \cdot C_l \cdot \lambda. \tag{4}$$

C_l indicates the price of the instance where the lth task is located. λ is a discount coefficient used to adjust the penalty for tasks that violate SLA.

In addition to cost, another factor to be considered is the utilization of resources. Here we mainly refer to the CPU utilization of VM instances, which can be calculated by the ratio of actual working time of CPU to the starting time of CPU in T_d. The utilization U can be expressed as follows:

$$U = \frac{T_u}{T_s}. \tag{5}$$

T_u is actual usage time of all CPUs in T_d, and T_s refers to the total CPU time of all started VM instances. T_s can be calculated as follows:

$$T_s = \sum_{i=0}^{n} Core^o_i \cdot CR^o_i \cdot T_d + \sum_{j=0}^{n} Core^r_j \cdot CR^r_j \cdot T_d, \tag{6}$$

where $Core^o_i$ is the number of cores of the ith on-demand instance and $Core^r_j$ is the number of cores of the jth reserved instance.

The goal of big data analysis service providers is to find the most appropriate VM renting plan to provision adequate services requested by their users. Such a plan should not only save the cost of renting VM instances, but also improve the utilization of resources as much as possible.

4. Reinforcement learning based auto-scaling approach

4.1. Theoretical foundations

Reinforcement Learning (RL) is one of the practical and mature paradigms and methodologies of machine learning. In contrast to other machine learning methods, RL explicitly considers the interaction between goal-directed agents and uncertain environment [21]. In RL, all agents have a clear goal, which is, to perceive all aspects of their environment, and choose actions according to their states to affect the environment they are interacting with. RL method is able to find a near optimal action sequence through the continuous interactions between the agent and the dynamic and stochastic environment in practical applications. Moreover, RL usually uses a trial and error approach to learn the optimal action sequence to obtain the most reward.

In order to solve the sequential decision making problem, we generally model it as a Markov Decision Process (MDP). MDP is a process which is used to simulate the stochastic strategy and reward process made by agents in a dynamic environment. MDP is built based on environments and agents, and can be described in a five tuple: $M =$

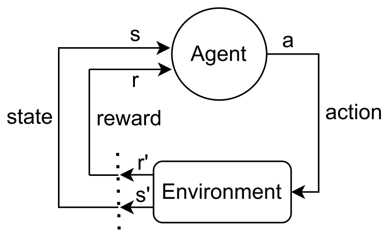


Fig. 2. Execution of MDP.

$(S,\ A,\ P,\ R,\ \gamma)$. S is a set of all possible environment states, and A is a set of actions taken by an agent. P is the probability that the environment state will change after the agent takes an action, which is also called state transition probability. a is an action in A. s and s' are states in S. R is an immediate reward obtained by the agent through taking action a in state s. $\gamma \in [0,1]$ indicates a discount factor, which is used to adjust the impact of future reward on current reward. The execution process of MDP can be simply described as shown in Fig. 2. The agent takes an action a in state s, which changes the environment and transfers state to s'. In the process of taking action a, the agent can receive a reward r. Then the agent will evaluate its behavior according to this reward so that it can make the most effective decision. The agent interacts with the environment continuously, and finally obtains a decision sequence that can maximize its long-term reward.

4.2. Reinforcement learning based auto-scaling approach

In big data analysis scenario, users' service requests arrive randomly and dynamically over time. However, in the long run, there is some regularity in a random environment, and so do the service requests of big data analysis. So we can view the formulation of cloud resource auto-scaling strategy as a sequential decision problem. By solving the sequential decision problem, we are able to obtain a scaling decision making sequence, which leads to an approximate prediction of real workload changes. There are many methods for solving sequential decision problems, such as model-based method, model-free method, etc. Temporal-Difference (TD) learning [22] is a most commonly used and effective one. TD uses a biased estimator to simulate long-term reward, which saves many intermediate steps. It uses only one step of random states and actions each time. Compared with other methods, TD converges faster and the variance of reward estimation s is smaller.

In practical applications, a method called Q-Learning based on TD is quite popular. Q-Learning is an algorithm that can obtain a near optimal decision sequence through the trial and error learning process without prior knowledge. Besides, Q-Learning operates as an off-policy algorithm [23], which means that it uses different methods to choose actions and evaluate strategies and leads to a faster convergence rate. This algorithm will maintain a Q table, which stores the accumulated reward $Q(s,a)$ that an agent obtains by taking action a when state is s. $Q(s,a)$ is also called Q value, and consists of the real-time reward obtained by the agent and the future long-term reward. Let r represent the real-time reward received by the agent taking action a in state s, and s' represent a new state of the environment after the agent taking action a. Let a' indicate the action that the agent can take in state s'. $\gamma \in [0,1]$ is a discount rate which is used to control the impact of long-term reward on current reward, and $\alpha \in [0,1]$ represents a learning rate, which is used to control the importance of newly learned knowledge. α and γ work together to adjust the update speed of Q-Learning, so as to reduce the error of learning results. Thus, the update formula of Q value is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Max_{a'} Q(s',a') - Q(s,a)]. \tag{7}$$

Q-Learning is suitable in finding the optimal decision sequence by trial and error mechanism in a stochastic environment. The auto-scaling problem is to find a cloud VM instance rental sequence that can meet the stochastic workload. Thus, it is a good choice to apply Q-Learning to solve auto-scaling problems. When using Q-Learning to solve problems, we need to first determine the decision cycle T_d, decision moment T_k ($k \in [1,2,\ldots]$) and episode. An episode usually refers to the period of time from a start state to an end state, or a fixed-length period of time. In the problem of auto-scaling in big data analysis scenario, it is quite difficult to choose a suitable start state and an end state, so we take the latter method to determine an episode by time. For the period duration each episode specifies, the agent makes multiple decisions and finally produces a decision sequence. In order to reflect the effectiveness of decisions, the decision cycle should be neither too long nor too short. We set the minimum cycle of on-demand VM instances as the decision cycle T_d.

In this section we also consider the state, action and reward related to the agent and environment, in addition to episode, decision cycle and decision moment.

4.2.1. States

A state is a description of the characteristics of the environment that an agent can observe. A good design of states can make the agent learn the best decision sequence faster. State space S contains various states that an agent may experience, and the agent will continue to transit between states until it reaches the termination state or the time constraint. In our design, we use a triple $(CRs,\ RT,\ WT)$ to describe the environmental state of the agent: $s = (CRs,\ RT,\ WT)$, where

- CRs is a vector that records the number of each instance rented by AaaS providers, as defined in Section 3;
- RT indicates the number of cycles that the tasks being executed in the last decision cycle need to run on the VM instances at the end of the last decision cycle;
- WT represents the number of cycles that tasks in the waiting state of the previous decision cycle need to run on the VM instances at the end of the previous decision cycle.

Moreover, in order to avoid the state dimension becoming exclusively large, which causes the final algorithm unable to converge, we set the upper and lower bounds for the number of cloud instances of each type in CRs. The number of instances to be added cannot exceed the upper bound of instances of the current type, and the number of instances to be released cannot be lower than the lower bound of instances of the current type.

4.2.2. Actions

Action is an operation selected by an agent to obtain the maximum reward according to the current environment state. Action space contains all actions that an agent can take, and it should be as simple and efficient as possible. In order to reduce the size of action space, we define an action as follows: $a = (V_1^o, \ldots, V_p^o, \ldots, V_n^o, V_1^r, \ldots, V_q^r, \ldots, V_n^r)$, where $p,\ q \in [1,2,\ldots,n]$. V_p^o indicates the number of the pth on-demand instance to be added in the next cycle. V_q^r is the number of the qth reserved instance to be added in the next cycle. Besides, we constrain each dimension value in action. For the on-demand instances, we set the number of each instance to be added to either 0 or 4. For the reserved instance, we set the corresponding number to either 0 or 8.

With regard to the actual use of cloud resources, we observe that the reserved instance is only released at the end of its life cycle. We also notice that the on-demand instance can be released at any time as long as it is idle. Therefore, we set the release of VM instances to automatic release, which means when an instance is idle and its life cycle ends, the instance can be automatically released. The size of action space has been reduced to a certain extent through the above design.

4.2.3. Reward function

Reward is used to measure the effectiveness of actions taken by an agent. A higher value of reward usually indicates that the action taken by the agent is more effective and more beneficial to the environment. At the same time, it also shows the decision made by the agent is closer to the optimal decision. In the auto-scaling problem of big data analysis scenario, AaaS providers usually focus on the cost of renting cloud resources and the utilization of resources. The utilization of resources is related to AaaS providers' cost, and improving resource utilization helps to reduce the amount of resources that AaaS providers need to rent. Due to the limited parallelization capacity of CPU, if AaaS providers focus only on reducing the waiting time of user requests, they need to lease more instances at the same time. In the long run, these additional leased resources are not fully utilized and will increase AaaS providers' cost. As a result, we take the total cost of AaaS providers and the CPU utilization of VM instances as the main parts of the reward function design.

Let C indicate the total cost of AaaS providers in a decision cycle, and U represent the CPU utilization of all cloud resources in the same decision cycle. At each decision moment t, we calculate the cost difference rate and CPU utilization difference rate from the initial time to time t and from time $t-1$ to time t respectively. The calculation formula is as follows:

$$\Delta C_{t\to 0} = \frac{C_0 - C_t}{C_0}, \quad \Delta C_{t\to t-1} = \frac{C_{t-1} - C_t}{C_{t-1}}; \tag{8}$$

$$\Delta U_{t\to 0} = \frac{U_t - U_0}{U_0}, \quad \Delta U_{t\to t-1} = \frac{U_t - U_{t-1}}{U_{t-1}}. \tag{9}$$

Among them, for the purpose of calculating reward conveniently, we uniformly set the positive value of the difference as the positive direction of the optimal target. We define the cost reward $Reward_C$ and the CPU utilization reward $Reward_U$ as follows:

$$Reward_C = \begin{cases} (e^{\Delta C_{t\to 0}} - 1) \cdot \frac{2}{1+e^{-\Delta C_{t\to t-1}}} \cdot bonus & \Delta C_{t\to 0} \geq 0 \\ (e^{-\Delta C_{t\to 0}} - 1) \cdot \frac{2}{1+e^{\Delta C_{t\to t-1}}} \cdot penalty & \Delta C_{t\to 0} < 0 \end{cases}, \tag{10}$$

$$Reward_U = \begin{cases} (e^{\Delta U_{t\to 0}} - 1) \cdot \frac{2}{1+e^{-\Delta U_{t\to t-1}}} \cdot bonus & \Delta U_{t\to 0} \geq 0 \\ (e^{-\Delta U_{t\to 0}} - 1) \cdot \frac{2}{1+e^{\Delta U_{t\to t-1}}} \cdot penalty & \Delta U_{t\to 0} < 0 \end{cases}, \tag{11}$$

where bonus is a base value of the reward received when the action taken can optimize the target, and penalty is a base value of the loss that the action taken makes it farther away from the final goal. The final reward $Reward$ is calculated by weighting the cost reward $Reward_C$ and CPU utilization $Reward_U$, which is expressed as follows:

$$Reward = \mu_1 \cdot Reward_C + \mu_2 \cdot Reward_U, \tag{12}$$

where μ_1 and μ_2 are two weights used to control the proportions of $Reward_C$ and $Reward_U$ in $Reward$, and $\mu_1 + \mu_2 = 1$.

4.2.4. Algorithm

Algorithm 1 presents our Q-Learning based auto-scaling method which makes cloud resource renting plans. The main steps of our algorithm are as follows. First, initialize the state space S, the action space A, learning rate α, and discount rate γ, and set the initial value of the Q table to 0 for every state–action pair, where state $s \in S$ and action $a \in A$. Second, the agent makes multiple action decisions for each episode. Every loop starts from initializing a state s, and then the agent chooses an action a in light of the state, Q table and ϵ-greedy strategy at each decision moment. After executing the action a, the state of environment will change from s to s'. If s' is not in state space S, it will be put in S and the items in Q table related to s' will be set to 0. Next, the agent will calculate the instant reward r by using Eqs. (8)–(12). Finally, the value in the Q table is updated through the calculation formula Eq. (7) of Q value. The algorithm will continue to loop as the above process until it reaches the upper limitation of episode we set in advance or the values in the Q table converge.

Algorithm 1: Q-Learning based Auto-Scaling Algorithm

```
Data: initial state space $S$, initial action space $A$, learning rate
      $\alpha$, discount rate $\gamma$
Result: decision sequence
1  $Q(s,a) \leftarrow 0, \forall s \in S, a \in A$;
2  for each episode do
3      Initialize $s$;
4      for each decision moment in episode do
5          choose an action $a$ using $\epsilon$ -greedy policy;
6          take action $a$;
7          move to new state $s'$;
8          calculate instant reward $r$ using Eqs. (8)–(12);
9          if $s'$ not in $S$ then
10             add $s'$ to $S$;
11             $Q(s',a') \leftarrow 0, \forall a' \in A$;
12         end
13         update $Q(s,a)$ using Eq. (7);
14         $s \leftarrow s'$;
15     end
16 end
```

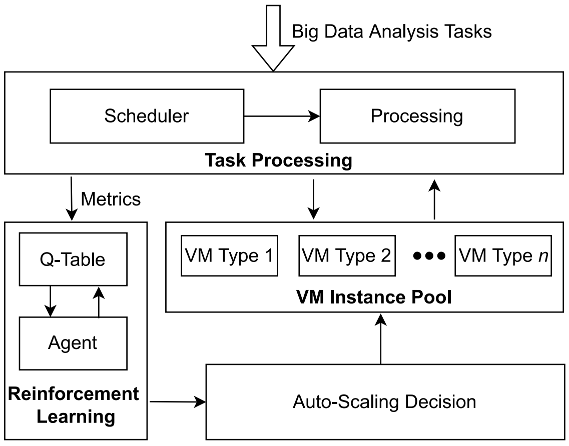
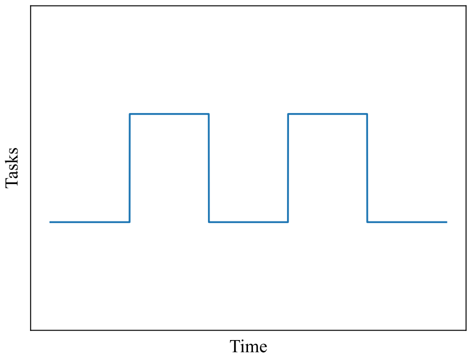


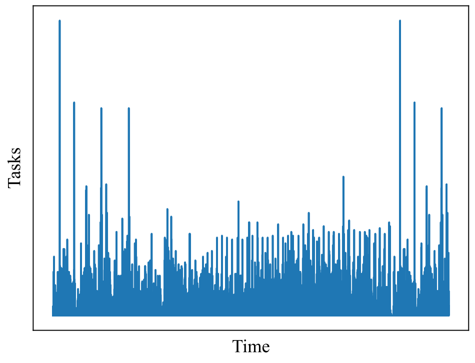
Fig. 3. Auto-scaling architecture of big data analysis scenario.

4.3. Auto-scaling architecture

As shown in Fig. 3, our auto-scaling architecture consists of VM instance pool, scheduler and reinforcement learning decision maker. VM instance pool is composed of multiple instances of different types, and instances of the same type have the same configuration (CPU and Memory). VM instances of different configurations have various concurrency processing capabilities. Reinforcement learning decision maker is mainly composed of Q table and an agent that makes scaling decisions. Users can submit their tasks at any time. When a task is submitted, the scheduler uses appropriate scheduling strategies (Best-Fit, FirstFit, etc.) to schedule it to a spare VM instance. Then when a decision point of reinforcement learning is reached, metrics such as cost and utilization generated in current cycle will be submitted to the agent. The agent selects an appropriate action according to Q table and alternating process of state/action in reinforcement learning. Finally, the selected action will work on the VM instance pool and change the number of VM instances of each type in VM instance pool. In this process, data and tasks are transmitted through high-speed network, and scheduling of the scheduler is completed within seconds. Thus, the time of task submission and task scheduling can be ignored.



(a) On-and-off workload



(b) Aliyun workload

Fig. 4. Workloads in on-and-off pattern and Aliyun cluster-usage traces.

5. Experimental design and analysis

In this section, we verify our proposed Q-Learning based auto-scaling method through a series of simulation experiments. We select several algorithms [18] to compare with our algorithm. First, we give a detailed introduction to the experimental environment settings, including the setting of experimental parameters, the selection of benchmark algorithms, etc. Then, we conduct experiments on two different workload modes. Finally, we compare the experimental results of different algorithms on the same performance metrics.

5.1. Experimental settings

We present the environment parameter settings and comparison algorithms involved in the experiments, and specify the algorithms used and some parameters in the generic environment. Besides, we have also made certain assumptions on the instance types, charging modes of cloud resources and workloads to better adapt to our experimental environment. We assume that there are two types of workloads: real trace and simulated trace. There are two types of instance for cloud resources, and there are two charging modes: reserved and on-demand. All the above is presented in this subsection.

Workload. We conduct our experiments on different workloads, each of which is a typical variation of user requests. As shown in Fig. 4, we consider two types of workload. Fig. 4(a) shows an on-and-off pattern, and the number of tasks submitted by users changes relatively smoothly and only changes between some values over time. Fig. 4(b) shows the cloud cluster workload of Alibaba company in China, which can be found in https://github.com/alibaba/clusterdata. The workload fluctuates greatly, but it also shows the load change from morning to night. We repeat and expand the data in proportion in order to find the hidden rules due to the lack of data.

Pricing. In order to match the real resource usages and prices as much as possible, we investigate cloud products from major cloud providers such as Amazon and Microsoft Azure, and finally select two VM instance types and two charging plans in Amazon EC2 for experiments. As shown in Table 1, the prices of the on-demand and reserved pricing plans of Amazon's two instance types are respectively P_1^o=0.0255\$/h, P_2^o=0.0510\$/h, P_1^r=0.0160\$/h, and P_2^r=0.0320\$/h. At the same time, since the real usage dataset of Aliyun cluster we acquired is a 24-hour dataset with the smallest time unit of seconds, we scale it up by converting the smallest time unit into minutes. So the original dataset is expanded 60 times from a day to a 60-day dataset. In addition, we also adjust the periods of on-demand instances and reserved instances to 1 h and 3 h respectively, which can reduce the inaccuracy caused by insufficient data on the experimental results. The time settings on the simulated data are the same as above.

Table 1
Pricing in Amazon EC2, as of Jul. 11, 2022.

Instance type	Configuration	Price of on-demand	Price of reserved
a1.medium	1v, 2G	0.0255$/h	0.0160$/h
a1.large	2v, 4G	0.0510$/h	0.0320$/h

Benchmark Algorithms. To measure the practical performance of our proposed algorithm, we compare it with several benchmark algorithms. Here we choose three benchmark algorithms, including the auto-scaling algorithm based on the random strategy, the algorithm based on the threshold, and the Q-learning based self-adaptive algorithm (QAA) that also uses reinforcement learning.

The auto-scaling algorithm based on the random strategy has the same state, reward, and action design as our proposed reinforcement learning algorithm. Details can be found in Section 5. The main difference between these two algorithms is the choice of action for each decision. In our algorithm, the agent selects the appropriate action according to the ϵ-greedy strategy and the state–action pair value in the Q table. While in the auto-scaling algorithm based on the random strategy, the action is chosen from a set that is the same as the action set in our algorithm with equal probability.

Threshold-based adjustment algorithm (TAA) is a commonly used decision-making algorithm. Unlike our algorithm, TAA usually sets a performance metric first. When the performance metric exceeds or falls below a threshold of θ, new operations will be performed. Besides, the performance metric and θ are related to the specific algorithm application scenario. In our experiment, we take the number of execution cycles on the configuration at the end of last decision cycle for tasks that are not completed as the performance metric. At the same time, we only consider the action of adding instances. In other words, if $(RT+WT)/T_s > \theta$, TAA will make decisions to rent extra VM instances in proportion. We set the number of extra instance rented each time to 6 for each charging mode and type of instance.

The QAA algorithm is an automatic scaling algorithm proposed in the literature [18] to make cloud resource scaling decisions for Software-as-a-Service (SaaS) providers. Both QAA and our algorithm adjust resources at the horizontal level, use reinforcement learning based methods and explore different charging modes for cloud resources. But QAA mainly solves the problem of resource scaling under dynamically changing user requests in web applications, which does not consider the processing time of each request. Moreover, it has a completely different design from our algorithm in terms of state, action, rewards, etc. In order to make QAA more effective in our scenario, we have made corresponding modifications to it, and the details are as follows:

Table 2
Experimental parameter settings.

Algorithm	Parameter settings
Random	λ=0.5 $\mu 1$=0.8, $\mu 2$=0.2
TAA	θ=0.31, λ=0.5 $\mu 1$=0.8, $\mu 2$=0.2
QAA	ϵ=0.1, λ=0.5 α=0.5, γ=0.5
Our Algorithm	ϵ=0.1, λ=0.5 α=0.5, γ=0.5 $\mu 1$=0.8, $\mu 2$=0.2

- State: Wei et al. combines all the types of instances currently owned in the environment and their corresponding numbers, the average workload of the last decision cycle, and the timestamp as a state. According to our scenario, we also use all types of instances currently in the environment and their corresponding numbers, the total number of tasks in the last decision cycle, and the timestamp as the state of the benchmark algorithm: $state = (sumTask,\ CRs,\ TIMESTAMP)$, where $sumTask$ indicates all the tasks in last decision cycle, and $TIMESTAMP$ is a time related execution process.
- Action: In the original literature, the authors take the number of instances that need to be increased or decreased in the next decision cycle as an action. The step of increases or decreases for each instance is 1. We retain the design of the action, but delete the part that reduces the number of instances and set the operation number to 6, in order to ensure the adaptability of this algorithm in our scenario as much as possible.
- Reward: In the QAA algorithm, its reward is calculated based on the profit of the SaaS providers and the resource utilization of renting VM instances. When an agent executes action a when the environment state is s, the environment state will become s'. The agent will receive a reward R, and $R(s', a) = profit(a) + performance(a)$. The $performance(a)$ is mainly determined by the processing capacity provided by the SaaS providers and the actual number of requests from users. In our experiments, we replace $profit(a)$ with −1 times cost and use the $performance(a)$ similar to that in the original literature. Details are as follows:

$$profit(a) = -C_t \tag{13}$$

$$performance(a) = \begin{cases} bonus & 0 \le \frac{T_s - T_u}{Td} \le 1 \\ bonus \cdot (1 - \frac{T_s - T_u}{T_s}) & \frac{T_s - T_u}{Td} > 1 \\ penalty \cdot (1 - \frac{T_s - T_u}{Ts}) & others \end{cases}. \tag{14}$$

Parameter Settings. The same algorithm will show different results under different parameter configurations. In order to avoid the influence of different parameters on the experimental results, we have unified the parameters of various algorithms used and some other necessary parameters. As shown in Table 2, the parameters involved in the algorithms in the experiments have been specified, where $\mu 1$ and $\mu 2$ are two weights, α indicates a learning rate, γ is a discount, and ϵ is a parameter that controls the rate of trial and error. λ is a discount coefficient used to adjust the penalty for tasks that violate SLA. θ will be used in the threshold-based adjustment algorithm. The specific meaning of parameters has been introduced in detail in Sections 4 and 5.

In addition to the above parameters used in the algorithm, some general parameters also need to be specified. Firstly, the decision-making cycle under the big data analysis service scenario needs to be determined. Because the dataset we obtained and the generated simulation data set are relatively small, we specify the decision-making cycle as 1 h and set the minimum unit of execution time of each task as 1 min. Then the number of decision cycles contained in an episode is dynamically set. In the simulation data set, we set it to 300, while in the real data set, we set it to 1000, ensuring that an episode can cover all tasks in the data set. Also, we set the number of various instances that the AaaS provider rents at the beginning, respectively as CR_1^o=30, CR_2^o=30, CR_1^r=30, and CR_2^r=30. We set the bonus and penalty in reward function to 2 and −1 respectively.

5.2. Algorithm convergence

In this part, we verify the convergence of our proposed scaling algorithm based on reinforcement learning in big data analysis scenario through experiments. It is necessary for AaaS providers to make the decision to adjust the type and number of instances they own at certain intervals, but it also requires significant manpower. In contrast, our approach automatically makes such adjustments at every decision moment. Fig. 5 shows the change curve of reward that the agent gains based on trial and error mechanism on two different workloads. Fig. 5(a) displays the performance result on on-and-off workload. From the figure, it can be observed that the decision made by the agent tends to be a random strategy, resulting in a low level of reward due to the lack of prior knowledge at the beginning. However, after 100 episodes, the reward value tends to stabilize, indicating that our algorithm has converged. Fig. 5(b) shows the performance of reward on the Aliyun workload. Compared with the former, we can clearly see that the algorithm's process is slower, because the real trace has higher randomness and the agent can learn less knowledge during training. But the value of reward finally reaches a stable value, which indicates that our algorithm can also achieve convergence on the real trace. Comparing the two results, we can find that the initial reward obtained by the agent in the real trace with a large amount of data will be about 4 times less than that obtained in the simulated trace. But the increase ratio of reward achieved in the end is similar. This shows that our algorithm can make good scaling decisions after a certain amount of learning, regardless of workloads.

5.3. Performance improvement on different workloads

In this part, we validate the performance of our proposed Q-Learning based auto-scaling method on different workloads. We mainly evaluate the algorithm from four aspects: reward improvement, cost saving, CPU utilization improvement and VM number reduction, and plot the improvement of the four aspects compared to the initial state as a line graph. Fig. 6 shows the performance of our algorithm on simulated workload and Aliyun workload. From the four graphs, we can find that our algorithm can achieve performance improvements on both simulated and real workloads. Compared to the initial state, AaaS providers can achieve 10% cost saving, nearly 50% reward improvement, 10%CPU utilization improvement and 10%reduction in VM number by using our algorithm. In addition, by performing on different workloads, we find that the impact of workload on the algorithm exists. For workloads with more stable fluctuations, our algorithm can achieve higher economic benefits and performance improvement. For workloads with higher randomness, the performance improvement that our algorithm can achieve will be relatively limited. In summary, compared to the initial state without any auto-scaling strategy, our algorithm can deliver significant cost saving and performance improvement at different time of different workloads.

5.4. Cost comparison

We first compare the performance of the proposed algorithm with several benchmark algorithms in terms of cost metric. In Section 3, we introduce that the cost of AaaS provider is composed of two components, i.e., VM cost and penalty cost which is caused by users' waiting

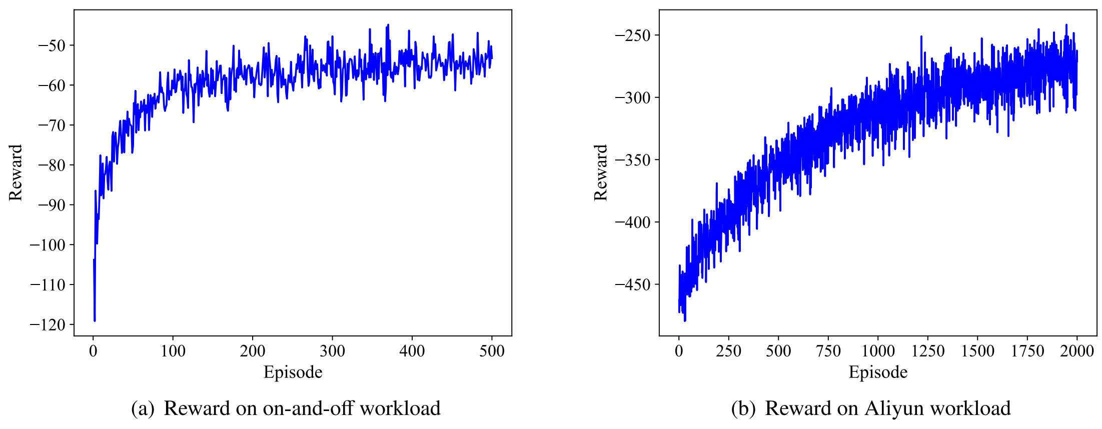


Fig. 5. Reward of our algorithm.

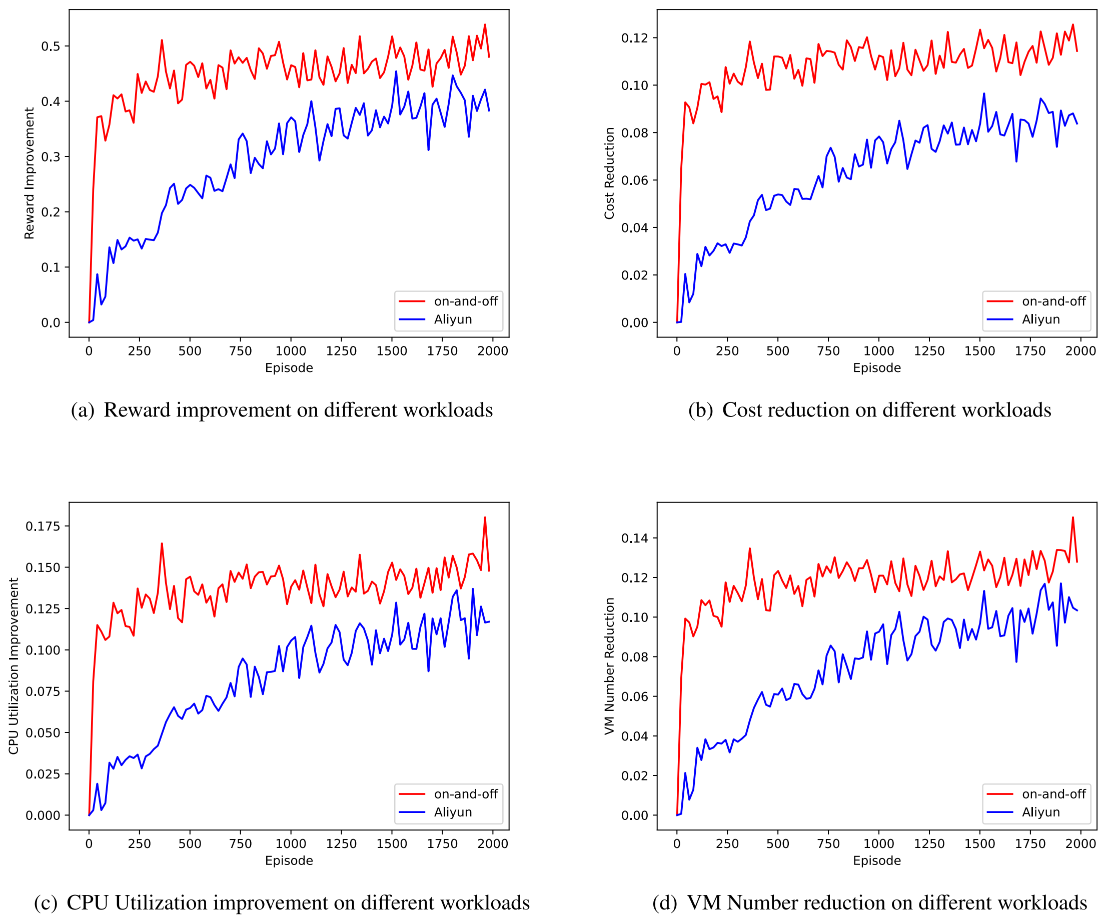
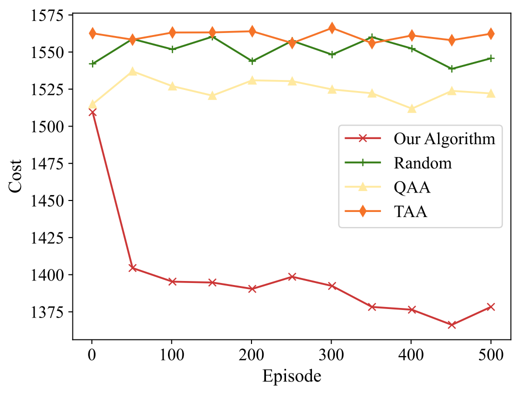


Fig. 6. Performance improvement of our algorithm.

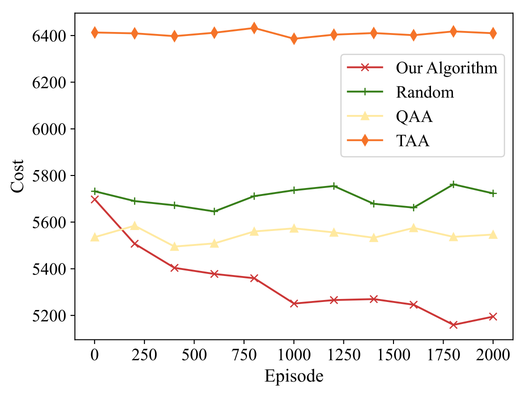
time violating SLA. Fig. 7 shows the variation of the impact of various algorithms on the total cost on two different workloads.

Each curve in the two graphs represents the impact of an auto-scaling algorithm on the cost of AaaS provider. Comparing the two graphs, we can clearly see that the decisions made by the Random algorithm are blind and random, and the cost keeps oscillating around a stable value regardless of the workload. In terms of cost control, TAA has hardly achieved the goal of reducing cost. TAA uses the number of idle instance as a threshold metric. When the number of idle instance exceeds the threshold, it will rent more instances of both on-demand and reserved patterns. However, this approach cannot handle the volatility of the workload, nor can it deal with the high stability of the workload. Therefore, TAA leads to the highest cost among the four methods. QAA is better than the first two algorithms in terms of cost, since the agent it employs has learned the variation of workloads to some extent by learning through iterations of multiple episodes. Compared with other algorithms, our algorithm preferentially uses reserved instances for workloads with small fluctuations and strong regularity, while on-demand instances are preferentially used for workloads with large fluctuations and strong randomness. Through the real-time adjustment of the agent, our algorithm shows the greatest cost savings in all workloads.

The performance of these algorithms in cost is usually related to the presence of learning mechanisms and the arrival pattern of the

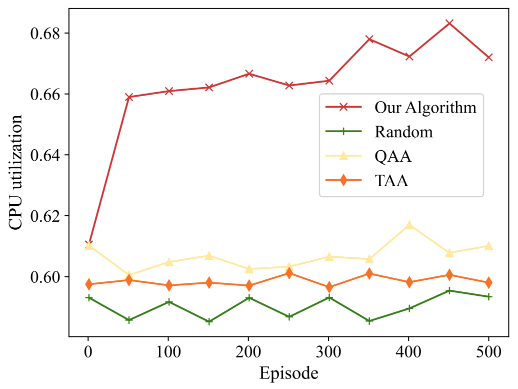


(a) Cost on on-and-off workload

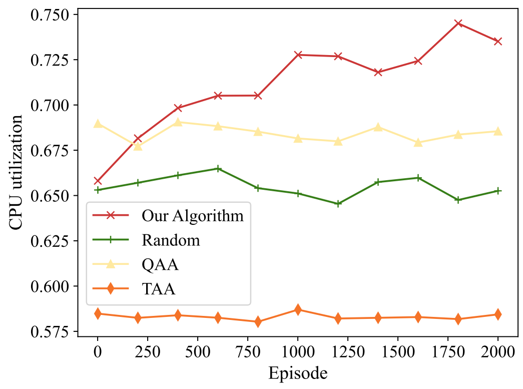


(b) Cost on Aliyun workload

Fig. 7. Cost of four algorithms.



(a) CPU utilization on on-and-off workload



(b) CPU utilization on Aliyun workload

Fig. 8. CPU utilization of four algorithms.

workload. From Fig. 7, it is evident that our proposed algorithm with a self-learning mechanism is able to effectively achieve cost control, which is reflected in the figure as a line with an arc. But the Random algorithm and TAA exhibit a shape close to a straight line, indicating a poor performance in cost control. For QAA algorithm, because the scene and algorithm do not match, the cost curve also appears close to a straight line. As for workload patterns, on the simulated dataset in on-and-off pattern, the agent in our proposed algorithm is able to learn the overall fluctuation of the workload quickly through the trial-and-error mechanism. But the agent takes longer time to learn due to the low regularity of the dataset on Aliyun workload. Concerning the cost of AaaS providers, our method shows a better performance than others.

5.5. *CPU utilization comparison*

In the previous subsection we compare the performance of different algorithms on cost, and in this part, we focus on analyzing the performance of these algorithms on resource utilization (CPU utilization). The results of the comparison are shown in Fig. 8, where Fig. 8(a) shows the results on on-and-off workload and Fig. 8(b) shows the results on Aliyun workload.

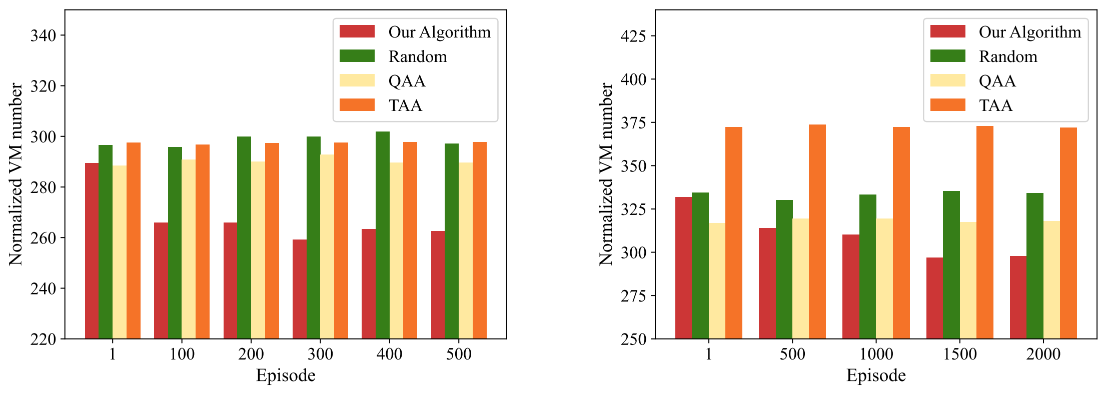
All four algorithms achieve a utilization of more than 58% of the CPU resources on on-and-off workload. From Fig. 8(a), we observe that at the beginning of the experiments, all methods make the AaaS provider's utilization of CPU approximately equal. However, as the number of iterations keeps increasing (i.e., the number of episodes increases), we can see that our proposed algorithm shows a more than 10% increase on CPU utilization than others, and finally reaches a stable value. This indicates that our algorithm successfully adapts to the current scenario and is able to make better decisions than the artificially formulated decisions.

In Aliyun workload, we also find a similar situation to the one described above, which is, the CPU utilization of the various algorithms is about equal at the beginning, and the difference on CPU utilization between the algorithms can be seen only after a period of iterations. Also, since the real workload is more random than the other simulated datasets, we can see in Fig. 8(b) that the convergence speed of our proposed algorithm is lower than that on on-and-off workload, and the improvement on CPU utilization is smaller than that on on-and-off workload. Besides, since the QAA algorithm is adapted from the scenarios in the original literature, it performs less well in the real data set and does not show significant convergence. For TAA and Random, we can find that their renting strategies cannot help AaaS providers improve the resource utilization due to the inappropriate scaling strategies.

All in all, our proposed algorithm performs best among these scaling methods in achieving the most improvement on CPU utilization. In other words, our method makes the most efficient scaling decisions.

5.6. *VM number comparison*

In our experiments, in addition to comparing the performance of the four algorithms on cost and resource utilization, we also analyze the number of VM instances rented by AaaS providers under different algorithms. In Section 5.1, we introduce Amazon VM instances involved in the experiments. The main difference between these two instances lies in the number of CPU core and memory. Therefore, in order to make the comparison obvious, we normalize the instances to those



(a) VM number on on-and-off workload (b) VM number on Aliyun workload

Fig. 9. VM number of four algorithms.

configured with 1 core and 2G memory in our experimental results. The experimental results are shown in Fig. 9.

Fig. 9(a) shows the performance of the four algorithms on simulation dataset. From this figure, we can see that our algorithm effectively controls the rental of VM instances by AaaS providers. Our algorithm reduces the number of resources rented by AaaS providers by about 13%. After all the algorithms converge, compared with the three algorithms, our method reduces the number of leased resources by 11.4%, 11.2%, and 9.3% respectively. Obviously, the results on simulation dataset show that our algorithm has excellent performance.

Similar to the experimental results on simulated dataset, our proposed algorithm also exhibits excellent performance on real dataset. Fig. 9(b) shows the results of three basic algorithms and our algorithm in controlling the number of VM instances on Aliyun dataset. Four algorithms use the same number of normalized instances at the beginning, but with the increase of episodes, our algorithm can gradually reduce the number of instances rented by AaaS providers. Finally, when the algorithms converge, our algorithm makes each episode use less resources than other algorithms and realizes the effective use of VM resources.

6. Conclusion and future work

The elasticity of cloud computing enables cloud users to flexibly choose charging modes and resources according to their actual demand. AaaS providers can optimize the cost and utilization of rented resources by utilizing the characteristics of different cloud resource charging modes. Faced with the uncertainty of the arrival of service requests, they need to continuously monitor the situation of request arrival to choose the most cost-effective resource scaling strategy. On this basis, we propose a novel Q-Learning based auto-scaling method to cope with unknown dynamic requests. This method learns knowledge from online service requests to minimize cost and maximize resource utilization. Through comparative experiments conducted on both real and simulated traces, our method achieves the utmost cost-effectiveness compared to the selected benchmark algorithms.

In the future, our focus will continue to be on auto-scaling problems in big data analysis scenario, but pay more attention to the collaboration between scaling and scheduling. In addition, we will enhance our algorithm and develop a more adaptive method combined with the knowledge of deep learning to deal with different types of workflows or tasks. Finally, we are also thinking about putting our work into industrial applications.

CRediT authorship contribution statement

Shihao Song: Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing – original draft, Writing – review & editing. **Li Pan:** Conceptualization, Methodology, Resources, Writing – review & editing, Visualization, Supervision. **Shijun Liu:** Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgment

The authors would like to acknowledge the support provided by the project ZR2022LZH018 supported by Shandong Provincial Natural Science Foundation, China.

References

[1] X. Zeng, S.K. Garg, M. Barika, A.Y. Zomaya, L. Wang, M. Villari, D. Chen, R. Ranjan, SLA management for big data analytical applications in clouds: A taxonomy study, ACM Comput. Surv. 53 (3) (2021) 46:1–46:40, http://dx.doi.org/10.1145/3383464, URL https://dl.acm.org/doi/10.1145/3383464.

[2] R. Kumar, A. Anand, P. Kumar, R.K. Kumar, Internet of things and social media: A review of literature and validation from twitter analytics, in: Proceedings of 2020 International Conference on Emerging Smart Computing and Informatics, (ESCI), IEEE, 2020, pp. 158–163, http://dx.doi.org/10.1109/ESCI48226.2020.9167558, URL https://ieeexplore.ieee.org/document/9167558.

[3] W. Voorsluys, J. Broberg, R. Buyya, Introduction to cloud computing, in: Cloud Computing, John Wiley & Sons, Ltd, 2011, pp. 1–41, http://dx.doi.org/10.1002/9780470940105.ch1, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470940105.ch1.

[4] N.S. Suhasini, S. Puli, Big data analytics in cloud computing, in: Proceedings of the 6th International Conference on Image Information Processing (ICIIP), Vol. 6, IEEE, 2021, pp. 320–325, http://dx.doi.org/10.1109/ICIIP53038.2021.9702705, URL https://ieeexplore.ieee.org/document/9702705.

[5] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, I. Ahmad, Cloud computing pricing models: A survey, Int. J. Grid. Distrib. 6 (5) (2013) 93–106, http://dx.doi.org/10.14257/ijgdc.2013.6.5.09, URL https://article.nadiapub.com/IJGDC/vol6_no5/9.pdf.

[6] Z. Gan, R. Lin, H. Zou, Adaptive auto-scaling in mobile edge computing: A deep reinforcement learning approach, in: Proceedings of the 2nd International Conference on Consumer Electronics and Computer Engineering, (ICCECE), IEEE, 2022, pp. 586–591, http://dx.doi.org/10.1109/ICCECE54139.2022.9712801, URL https://ieeexplore.ieee.org/document/9712801.

[7] R. Han, L. Guo, M. Ghanem, Y. Guo, Lightweight resource scaling for cloud applications, in: Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGRID), IEEE, 2012, pp. 644–651, http://dx.doi.org/10.1109/CCGrid.2012.52, URL https://ieeexplore.ieee.org/document/6217477.

[8] W. Wang, B. Liang, B. Li, Optimal online multi-instance acquisition in IaaS clouds, IEEE T. Parall. Distr. 26 (12) (2015) 3407–3419, http://dx.doi.org/10.1109/TPDS.2014.2385697, URL https://ieeexplore.ieee.org/document/6995955.

[9] W. Lin, J.Z. Wang, C. Liang, D. Qi, A threshold-based dynamic resource allocation scheme for cloud computing, Proc. Eng. 23 (2011) 695–703, http://dx.doi.org/10.1016/j.proeng.2011.11.2568, PEEA 2011, URL https://www.sciencedirect.com/science/article/pii/S1877705811054117.

[10] C. Yang, Q. Huang, Z. Li, K. Liu, F. Hu, Big data and cloud computing: innovation opportunities and challenges, Int. J. Digit. Earth 10 (1) (2017) 13–53, http://dx.doi.org/10.1080/17538947.2016.1239771, URL https://www.tandfonline.com/doi/full/10.1080/17538947.2016.1239771.

[11] M. Mao, M. Humphrey, Scaling and scheduling to maximize application performance within budget constraints in cloud workflows, in: Proceedings of the 27th IEEE International Symposium on Parallel and Distributed Processing, (IPDPS), IEEE, 2013, pp. 67–78, http://dx.doi.org/10.1109/IPDPS.2013.61, URL https://ieeexplore.ieee.org/document/6569801.

[12] R. Jannapureddy, Q.-T. Vien, P. Shah, R. Trestian, An auto-scaling framework for analyzing big data in the cloud environment, Appl. Sci. 9 (7) (2019) 1417, http://dx.doi.org/10.3390/app9071417, URL https://www.mdpi.com/2076-3417/9/7/1417.

[13] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, (SC), Association for Computing Machinery, New York, NY, USA, 2011, pp. 49:1–49:12, http://dx.doi.org/10.1145/2063384.2063449.

[14] W. Si, L. Pan, S. Liu, A cost-driven online auto-scaling algorithm for web applications in cloud environments, Knowl.-Based Syst. 244 (2022) 108523, http://dx.doi.org/10.1016/j.knosys.2022.108523, URL https://www.sciencedirect.com/science/article/pii/S095070512200226X.

[15] Y. Gari, D.A. Monge, C. Mateos, C.G. Garino, Learning budget assignment policies for autoscaling scientific workflows in the cloud, Clust. Comput. 23 (1) (2020) 87–105, http://dx.doi.org/10.1007/s10586-018-02902-0, URL https://link.springer.com/article/10.1007/s10586-018-02902-0.

[16] I. George Fernandez, J. Arokia Renjith, A novel approach on auto-scaling for resource scheduling using AWS, in: Proceedings of International Virtual Conference on Industry 4.0, (IVCI4.0), Springer Singapore, Singapore, 2021, pp. 99–109, http://dx.doi.org/10.1007/978-981-16-1244-2_8, URL https://link.springer.com/chapter/10.1007/978-981-16-1244-2_8.

[17] S. Horovitz, Y. Arian, Efficient cloud auto-scaling with SLA objective using Q-Learning, in: Proceedings of the 6th IEEE International Conference on Future Internet of Things and Cloud, (FiCloud), IEEE, 2018, pp. 85–92, http://dx.doi.org/10.1109/FiCloud.2018.00020, URL https://ieeexplore.ieee.org/document/8457997.

[18] Y. Wei, D. Kudenko, S. Liu, L. Pan, L. Wu, X. Meng, A reinforcement learning based auto-scaling approach for SaaS providers in dynamic cloud environment, Math. Probl. Eng. 2019 (2019) http://dx.doi.org/10.1155/2019/5080647, URL https://www.hindawi.com/journals/mpe/2019/5080647/.

[19] O. Runsewe, N. Samaan, Cloud resource scaling for big data streaming applications using a layered multi-dimensional hidden markov model, in: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGRID), 2017, pp. 848–857, http://dx.doi.org/10.1109/CCGRID.2017.147, URL https://ieeexplore.ieee.org/document/7973790.

[20] Amazon EC2 pricing, 2022, https://aws.amazon.com/cn/ec2/pricing/, (Accessed on 30 May 2022).

[21] R. Sutton, A. Barto, Reinforcement learning: An introduction, IEEE Trans. Neural Netw. 9 (5) (1998) 1054, http://dx.doi.org/10.1109/TNN.1998.712192, URL https://ieeexplore.ieee.org/document/712192.

[22] Y. Wan, A. Naik, R.S. Sutton, Learning and planning in average-reward markov decision processes, in: Proceedings of the 38th International Conference on Machine Learning (ICML), Vol. 139, PMLR, 2021, pp. 10653–10662, http://dx.doi.org/10.48550/arXiv.2006.16318, URL https://proceedings.mlr.press/v139/wan21a.html.

[23] R. Fakoor, P. Chaudhari, S. Soatto, A.J. Smola, Meta-q-learning, 2019, http://dx.doi.org/10.48550/arXiv.1910.00125, arXiv preprint arXiv:1910.00125, arXiv:1910.00125, URL https://arxiv.org/abs/1910.00125.

Shihao Song received the B.S. degree in Software Engineering from Shandong University in 2021. He is currently working toward the M.S. degree in Software Engineering with the School of software, Shandong University, Jinan, China. His research interests include cloud computing and service computing.

Li Pan received the Ph.D. degree from the Research Center of Human-Computer Interaction Virtual Reality, Shandong University. She is currently an Associate Professor with the School of Software, Shandong University, and a member of the HCIVR Group. Her main research interests include service computing and cloud computing.

Shijun Liu received the B.S. degree in oceanography from the Ocean University of China, and the M.S. and Ph.D. degrees in computer science from Shandong University, China. He is currently a Professor with Shandong University. His current research interests include services computing, enterprise services computing, and services system for manufacturing.