A Deep Reinforcement Learning Based Resource Autonomic Provisioning Approach for Cloud Services

Qing Zong[1], Xiangwei Zheng[1(✉)], Yi Wei[1(✉)], and Hongfeng Sun[2]

[1] School of Information Science and Engineering, Shandong Normal University, Jinan, China
zongqing_sdnu@163.com, xwzhengcn@163.com, weiyi@sdnu.edu.cn
[2] School of Data and Computer Science, Shandong Women's University, Jinan, China
nameshf@163.com

Abstract. Resource elastic scheduling is a key feature of cloud services. The elastic makes cloud services have the ability to flexibly increase or decrease resources to satisfy user needs, and dynamically allocate resources for cloud services on demand. The amount of resources to be configured is determined at runtime based on the changes in workload to flexibly respond to the fluctuating demands of cloud services. Appropriate resources need to be configured in advance. In this article, we propose a dynamic resource provisioning framework based on the MAPE loop, and use a two-tier elastic resource configuration for collaborative work. In order to implement the proposed framework, we propose an elastic resource scheduling algorithm based on a combination of the autonomic computing and deep reinforcement learning (DRL) to reduce task rejection rate of the virtual machine (VM) and increase utilization to obtain as much profit as possible. In this paper, Experimental results using actual Google cluster tracking results show that the proposed policy reduces the total cost about 17%–58% and increases the profit by up to not less than 9%, reduces the service level agreement (SLA) violations to less than 0.4% to better guarantee the quality of service (QoS).

Keywords: Cloud computing · DRL · Resource scheduling · Autonomic computing

1 Introduction

Cloud computing has emerged as the most popular commerce computing model in today's computer industry. Cloud computing can provide users with computing resources such as computing power, storage, and bandwidth as needed. Cloud applications are software products that run on cloud environments. Generally, each cloud application consists of one or more cloud services. According to different service patterns that users need, cloud computing can be divided into three types: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In this paper, we consider a common three-tier cloud environment which involves three roles: IaaS vendors, SaaS

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 350, pp. 132–153, 2021.
https://doi.org/10.1007/978-3-030-67540-0_8

providers, and users. IaaS vendors provide their customers access to various computing resources, such as storage, servers and networking. SaaS providers rent several VM instances from IaaS vendors to construct their cloud services, and users purchase cloud services from SaaS providers.

The elastic characteristics of cloud computing enable SaaS providers to adapt to the changes in the workload of their cloud services by dynamically configuring and reallocating resources. Ideally, the available resources are as close as possible to current needs at each time point. However, user requests are potentially uncertain. It is not easy to determine the appropriate amount of resources for cloud services during execution. On the one hand, if the number of resources provided by the SaaS provider is greater than the demand of user requests, it will cause the waste of resources and unnecessary monetary cost. On the other hand, if the number of resources provided by the SaaS provider is less than the amount of user requests, the under-provisioned situation may result in lower revenue and missing potential customers.

In order to deal with the above resource supply problem of cloud applications, dynamic resource supply is used. Dynamic resource supply is an effective method, and its basic idea is to supply resources based on changes in the workload of cloud applications. The goal is to automate the dynamic configuration of resources by minimizing the cost of renting resources from IaaS providers and meeting users' SLA. For SaaS providers, they concern how to maximize their profits and guarantee customer satisfaction during the execution of their cloud applications.

In this paper, we combine the concepts of autonomous computing and deep reinforcement learning (DRL), and propose a cloud elastic resource scheduling algorithm. IBM has proposed a MAPE (Monitoring, Analysis, Planning, and Execution) control loop [1], which contains four stages: monitoring (M), analysis (A), planning (P), and execution (E). We use a two-layer MAPE control loop to better allocate resources for SaaS providers. The first MAPE control loop is in charge of renting appropriate virtual machines from the IaaS provider. The second MAPE control loop is responsible for resource sharing and coordination among cloud services. DRL is an important field of machine learning. It can simultaneously give play to the representational advantages of deep learning and the decision-making advantages of reinforcement learning. In the first MAPE control loop, we use DRL as the decision-making agents that interact repeatedly with the cloud environment and make resource adjustment decisions automatically. The main contributions of this paper can be summarized as follows:

- We design a dynamic resource supplying system, which consists of two sets of the MAPE loop to adjust the supplied resources for cloud services or balance resources among cloud services.
- We use DRL as the decision-making agents to improve the performance of the resource dynamic provisioning. At the same time, multiple cloud services learn collaboratively, which effectively accelerates the learning speed.
- For the elastic scaling of cloud resources, we adopt a vertical and horizontal hybrid method to adjust cloud resources elastically, which has a higher utilization rate and a lower SLA violation rate.
- We use real workload tracking to conduct experiments under different metrics to evaluate the performance of our approach.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 formulates the problem and proposes a formal description. The necessary theoretical background is presented in Sect. 4. In Sect. 5, we propose a dynamic resource supplying framework and the corresponding elastic resource scheduling algorithm. Section 6 presents an evaluation and discusses the experimental results. Section 7 concludes our work and provides future research directions.

2 Related Works

At present, many scholars have done a lot of research on the resource provisioning in the cloud environment [1–3]. Traditional resource provisioning techniques employ predefined policies to guide application scaling, and manually specify scaling rules after an application is deployed. These rules specify a pair of thresholds and change the number of VMs after triggering the expansion condition [4, 5]. However, the threshold approach adds or reduces fixed numbers of VM instances at a certain time, which may not provide suitable resources in time when workload changes. In order to scale the amount of resource properly, it is necessary to predict workloads of cloud services for better performance. Prediction algorithm can allocate resources in advance, but the fluctuated amount of user requests will bring uncertain error [6–8].

With the development of the theory of machine learning, machine learning methods become more and more attractive in the field of cloud computing. Mazidi et al. [9] propose a new method based on the K-nearest neighbor algorithm that is used to analyze and label virtual machines, and statistical methods are used to make expansion decisions. Wei et al. [10] propose 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. Ghobaei et al. [11] propose a cloud service hybrid resource supply method based on the concepts of autonomous computing and reinforcement learning (RL), and an autonomous resource supply framework inspired by the cloud layer model. Li et al. [12] present the resource controllers based on statistical machine learning which execute on different VMs in cloud environments to achieve application service level objects (SLOs) under fluctuating time-varying workloads and unpredictable variations of system situations. Gill et al. [13] propose a based particle swarm optimization resource scheduling approach which is used to execute workloads effectively on available resources. Salah et al. [14] present an analytical model based on Markov chains to predict the number of or VM instances needed to satisfy a given SLO performance requirement. In our paper, we use a shared resource pool to reduce prediction errors, and DRL-based agents that interact repeatedly with the cloud environment to make better decisions.

3 Problem Statement

We consider a common three-tier cloud environment which involves three roles: IaaS vendors, SaaS providers, and users. The user submits tasks to the SaaS provider for execution. The SaaS provider rents a certain number of VMs from the IaaS provider and deploys cloud services to complete the requests submitted by users. IaaS providers

provide almost unlimited resources in the form of virtual machines. The number of user requests is continuous and fluctuating. The main goal of SaaS providers is to maximize profits. Therefore, SaaS providers need to rent as few resources as possible and respond flexibly to workload fluctuations in order to minimize the cost of VM rental and the penalty caused by SLA violations. However, deciding the suitable amount of VMs for cloud services during execution is quite difficult.

Notions and variables used throughout the paper are defined in Table 1. In the cloud environment, the IaaS provider provides M types of VMs. The price of each VM type m is $VPrice_m$. The SaaS provider provides I types of cloud services. Each cloud service is composed of different types of VMs and the service fee is $CPrice_i$. Cloud services serve user requests continuously. Each user request $\mathrm{Req}_{\mathrm{u}}^r$ contains the arrival time AT_u^r, the running time $RTime_u^r$ and the deadline time DL_u^r.

An SLA violation occurs when the SaaS provider fails to guarantee the predefined user SLA. FT_u^r and DL_u^r are the finishing time and deadline time for the user request Req_u^r. If the finishing time is greater than the deadline, an SLA violation will occur. The SLA is defined as:

$$SLA\left(Req_u^r\right) = \begin{cases} Yes(1)\ FT_u^r - DL_u^r > 0 \\ No(0)\ \ otherwise \end{cases} \tag{1}$$

The Total cost is the cost consumed by the SaaS provider to satisfy all cloud services. It includes the VM Costs and the Compensation, and it can be described as:

$$Total\ Cost = VM\ \ Costs + Compensation \tag{2}$$

VM cost is the total cost of all VMs rented from the IaaS provider:

$$VM\ \ Costs = \sum\nolimits_{n=1}^{N} VM\ \ cost_n \tag{3}$$

For $VM\ \ cost_n$, it depends on the VM price $VPrice_m$ and the initiation VM price $init\ price_m$ of type m, and the duration of time for which the VM is on $VTime_i$, which is calculated as:

$$VM\ \ Cost_n = (VPrice_m \times VTime_n) + init\ price_m \quad \forall n \in N; m \in M \tag{4}$$

Compensation is the total penalty cost for all user requests, and it is expressed as:

$$Compensation = \sum_{u=1}^{U} \sum_{r=1}^{R_u} Penalty\left(Req_u^r\right) \tag{5}$$

For each user request Req_u^r, the *Penalty* is similar to related work [15] and is modeled as a Linear function can be described as:

$$Penalty\left(Req_u^r\right) = \lambda_u^r \times \frac{FT_u^r - DL_u^r}{\Delta t} \tag{6}$$

where λ_u^r is the penalty rate for the failed request which depends on the request type, and Δt is a fixed time interval.

The purpose of this paper is to enable SaaS providers to minimize the payment for using resources from IaaS provider, and to guarantee the QoS requirements of users.

Table 1. Notations and definitions.

Notation	Definition
U	Number of users
Use_u^r	The u^{th} user
C	Total number of requests for all users
Ru	The number of requests belongs to u^{th} user
Req_u^r	r^{th} request of the u^{th} user
$RTime_u^r$	The running time of the r^{th} request of the u^{th} user
AT_u^r	The time at which the r^{th} request belongs to u^{th} user arrival system
ST_u^r	The start time of the user request Req_u^r
FT_u^r	The finish time of the user request Req_u^r
DL_u^r	The deadline time of the user request Req_u^r
$Cloud_i$	i^{th} cloud service offered by SaaS provider
$CPrice_i$	i^{th} cloud service of cloud service charges
$NumVM_i(\Delta t)$	Number of VMs allocated to the $Cloud_i$ at the Δt^{th} interval
$Num_i(\Delta t)$	Number of CPU resources allocated to the VM of the $Cloud_i$ at the Δt^{th} interval
$Remain_i(\Delta t)$	Number of remaining available CPU resources for the VM of the $Cloud_i$ at the Δt^{th} interval
$NumReq_i(\Delta t)$	Number of requests for the $Cloud_i$ at the Δt^{th} interval
$AvgReq_i$	Average (Ave) number of requests for the $Cloud_i$
$NumRun_i(\Delta t)$	The number of requests being executed by the $Cloud_i$ at the Δt^{th} interval
$NumWait_i(\Delta t)$	The number of requests blocked by the $Cloud_i$ at the Δt^{th} interval
$Utili_i(\Delta t)$	The CPU Utilization of the $Cloud_i$ at the Δt^{th} interval
N	Number of initiated VMs
M	Number of VM types
I	The total number of cloud services offered by the SaaS provider
$VPrice_m$	Price of VM type m
$init\ price_m$	The initiation VM price of type m
$VTime_i$	The duration of time for the n^{th} VM is on
$RTime_u^r$	The running time of the r^{th} request of the u^{th} user

4 Theoretical Background

This section exposes a brief overview of the autonomic computing and deep reinforcement learning techniques.

4.1 Autonomic Computing

To achieve autonomic computing, IBM has suggested a reference model for auto-nomic control loops, which is called the MAPE (Monitor, Analysis, Plan, Execute, Knowledge) loop and is depicted in Fig. 1. The intelligent agent perceives its environment through sensors and uses the information to control effectors carry out changes to the managed element [16].

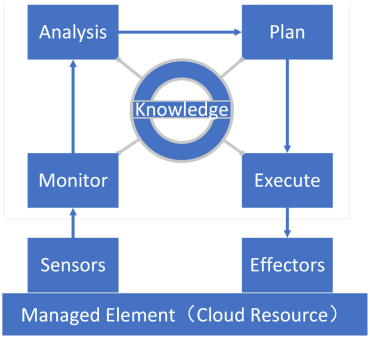

Fig. 1. The MAPE control loop

In the cloud environment, the MAPE loop can be used to manage cloud resources dynamically and automatically. The managed element represents cloud resources. Monitor collects CPU, RAM and other related information from managed element through sensors. The analysis phase analyzes the data received in the monitor phase in accordance with the prescribed knowledge. The plan stage processes the analyzed data and makes a certain action plan. In the execute phase, the action requirements are reflected to the effectors to execute on the managed element. Knowledge is used to control the shared data in the MAPE-K cycle, constrain each stage and adjust the configuration according to the indicators reported by the sensors in the connected environment.

4.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) [17] is an important field of machine learning. It has the representational advantages of deep learning and the decision-making advantages of reinforcement learning, providing the possibility to solve more complex large-scale decision-making control tasks. The general framework of deep reinforcement learning is shown in Fig. 2. The agent uses a deep neural network to represent the value function, strategy function or model, and the output of deep learning is the agent's action a. Next, the agent obtains feedback reward r by performing action a in the environment, and takes r as the parameter of the loss function. Subsequently, the loss function is derived based on the stochastic gradient algorithm. Finally, the weight parameters in the deep neural network are optimized after the training of the network model.

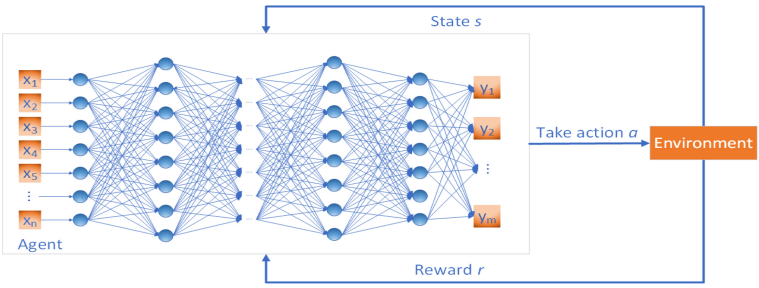

Fig. 2. Reinforcement learning with DNN

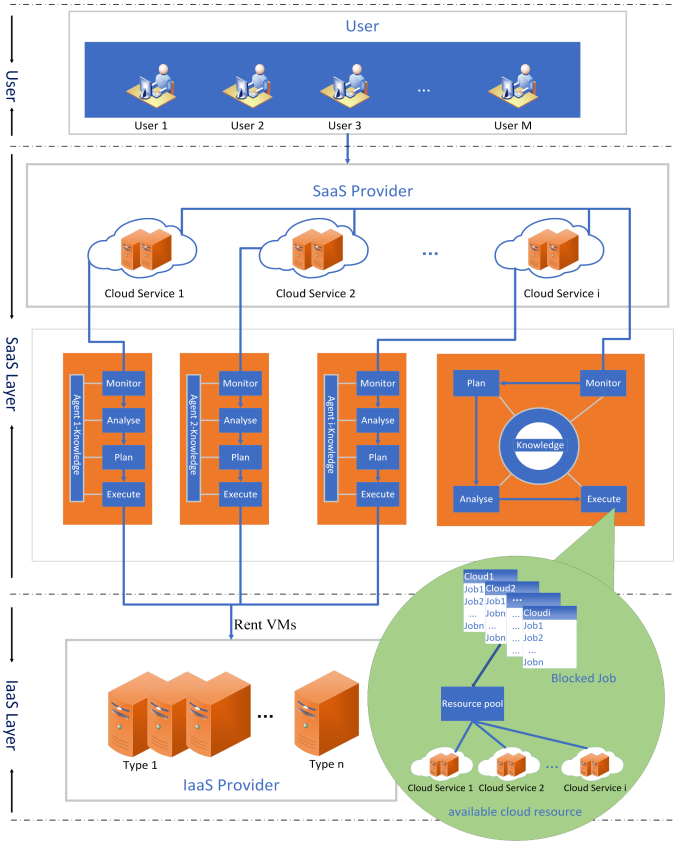

Fig. 3. Resource provisioning framework based on the control MAPE loop.

5 Framework and Algorithm

In this section, we propose a dynamic resource provisioning framework based on the MAPE loop, and describe in detail the elastic resource scheduling algorithm used to control the MAPE loop.

5.1 The Dynamic Resource Provisioning Framework

In order to better rent and manage VM resources, we propose a dynamic resource provisioning framework for the SaaS provider, which is shown in Fig. 3. With the support of the MAPE loop, each cloud service uses predictive technology and DRL technology to rent the appropriate number of VMs from IaaS provider. All this happens in parallel and cloud services do not interfere with each other. When all cloud services complete the MAPE loop, another MAPE cycle will be started to adjust the resource waste caused by prediction errors. The sensors collect resource waste of cloud service and user request blocking. The MAPE loop implements appropriate strategies to share resources between cloud services. The effector creates a resource pool based on the decision of the MAPE loop, and transfers blocking tasks to resources for execution. After deploying the SaaS provider with various cloud services in the dynamic resource provisioning framework, the MAPE loop runs until there are no cloud services running in the SaaS provider.

5.2 Elastic Resource Scheduling Algorithm

In this section, we will explain our elastic resource scheduling algorithm for cloud services in more detail. The algorithm consists of two groups of control MAPE loops. The first group of MAPE includes Monitoring Phase A, Analysis Phase A, Planning Phase A, Execution Phase A. It adjusts the number of VMs in advance by predictive technology and DRL algorithm to achieve horizontal scaling. Because new machines generally take between 5 and 10 min to be operational, we set it loop every five minutes. The second group of MAPE includes Monitoring Phase B, Analysis Phase B, Planning Phase B, Execution Phase B. It loops every one minute, and achieve vertical scaling by resource sharing, collaborative work and self-adapting management. The proposed algorithm (see Algorithm 1) will run until the SaaS provider does not receive a new user request.

Algorithm 1: Pseudo code for Autonomic Resource Provisioning

```
1: begin
2: for k := 0 to K do // k is a time of counter and the number of one-minute intervals is K,
3:    if k ≡ 0 mod 5 then
4:       for (every cloud service Cloud_i in SaaS Provider at the interval Δt) do
5:          Monitoring Phase A (Cloud_i)
6:          Analysis Phase A (Cloud_i)
7:          Planning Phase A (Cloud_i)
8:          Execution Phase A (Cloud_i)
9:       end for
10:   end if
11:   Monitoring Phase B
12:   Analysis Phase B
13:   Planning Phase B
14:   Execution Phase B
15: end for
```

(1) Monitoring Phase

In monitoring phase A, for each cloud service $Cloud_i$ provided by the SaaS provider, the monitor will collect the number of VMs $NumVM_i(\Delta t)$ that the SaaS provider rents from the IaaS provider, and the user's request for the cloud service Cloudi, the number of requests $NumReq_i(\Delta t)$ and the number of requests that the cloud service Cloudi has not started $NumWait_i(\Delta t)$. At the same time, the monitor will also detect the CPU utilization of the VM of the cloud service Cloudi $Utili_i(\Delta t)$. In the monitoring phase B, for each cloud service Cloudi provided by the SaaS provider, the monitor collects the remaining resources of the VM leased by each cloud service $NumSur_i(\Delta t)$ and the number of requests for the cloud service Cloudi that have not started $NumWait_i(\Delta t)$. All the information obtained in the monitoring phase is stored in the database and will be used in other phases.

(2) Analysis Phase

The analysis phase A obtains the current number of user requests $NumReq_i(\Delta t)$ for the cloud service $Cloud_i$ from the monitoring phase A, in order to predict the number of requests $NumReq_i(\Delta(t+1))$ for cloud service $Cloud_i$ in the next interval $\Delta(t+1)$. In this paper, we use the Autoregressive Integrated Moving Average Model (ARIMA) [18] to predict future demands for cloud service $Cloud_i$, as shown in Algorithm 2.

ARIMA (p, d, q) model first uses the difference method for the historical data X_t of the non-stationary user request, and performs the smoothing preprocessing on the sequence to obtain a new stationary sequence $\{Z_1, Z_2, \ldots, Z_{t-d}\}$. The difference method is expressed as:

$$\Delta y_{X_t} = y_{X_{t+1}} - y_{X_t} \quad t \in Z \tag{7}$$

We then fit the $ARMA(p, q)$ model, and restore the original d-time difference to get the predicted data Y_t of X_t. Among them, the general expression of $ARMA(p, q)$ is:

$$X_t = \varphi_1 X_{t-1} + \cdots + \varphi_1 X_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-q} \quad t \in Z \tag{8}$$

In the formula, the first half is the autoregressive part, the non-negative integer p is the autoregressive order, φ_1, …, φ_p is the autoregressive coefficient. The second half is the moving average part, and the non-negative integer q is the moving average order, θ_1, …, θ_q is the moving average coefficient. X_t is the related sequence of user request data, ε_t is the sequence of independent and identically distributed random variables, and satisfies $V\ ar\epsilon_t = \sigma_\varepsilon^2 > 0$.

Algorithm 2: Pseudo code for Analysis Phase A ($Cloud_i$)

1: **Input:** The number of requests for cloud service $Cloud_i$
2: **Output**: Prediction value ($Cloud_i$)//Y_t
3: **begin**
4: $\{X_1,X_2,\ldots,X_t\}$←History ($Cloud_i$) // History of the number of request for service $Cloud_i$
5: $\{Z_1, Z_2,\ldots,Z_{t\text{-}d}\}$←make the tranquilization of $\{X_1,X_2,\ldots,X_t\}$ using Eq.(7)
6: prediction value($Cloud_i$)←Calculate X_t based on $\{Z_1, Z_2,\ldots,Z_{t\text{-}d}\}$ using Eq.(8)
7: **return** prediction value ($Cloud_i$)
8: **end**

Analysis phase B selects cloud services according to certain criteria for resource transfer in subsequent stages.

(3) Planning Phase

In planning phase A, we use a DRL-based method to make decisions to achieve horizontal scaling of cloud services and divide the entire process into three states: normal-resources, lack-resources, and waste-resources. According to algorithm 3, firstly, we obtain the predicted value of the number of tasks requested in the next interval from the analysis stage and calculate the cloud service $Cloud_i$ resource at time t to meet the CPU load requested by all tasks at next time (line 2), and then determine the current state based on it (line 3–5). We randomly choose an action with the probability of ε, otherwise choose the action with the largest Q value in the network (line 8–9). Then we perform the action a_t, boot users to request access to cloud services and observe reward r_t (line10–12). We store pair (S_k, A_k, R_k, S_{k+1}) as knowledge D (line 13), and take some pairs out of the knowledge D to optimize the parameters of the network θ.

Algorithm 3: Pseudo code for Planning Phase A($Cloud_i$)

Input: Prediction value (Y_t),
The number of requests blocked ($NumWait_i(\triangle t)$)
The number of requests being executed ($NumRun_i(\triangle t)$)
The number of CPU resources allocated to the VM ($Num_i(\triangle t)$)
The time of counter k

Output: selected action ($Cloud_i$) = a_k

Initialization: Network parameters; $U_{upper-threshold}$=1.0; $U_{lower-threshold}$=0.25

1: **begin**
2: Calculate the utilization ($\text{Utili}_i(\triangle)$) of the latest VM using Eq.9
3: **if** $Utili_i(\triangle t) > U_{upper-threshold}$ **then** predict state = lack-resources
4: **else if** $Utili_i(\triangle t) < U_{upper-threshold}$ **then** predict state = waste-resources
5: **else** predict state = normal- resources
6: **end if**
7: s_k = predict state
8: with probability ε select a random action a_k
9: otherwise select $a_k = argmax_a Q(\phi(s_t),a,\theta)$
10: executes action a_t
11: boot users to request access to cloud services
12: observe reward r_t ,and observe state transition at next decision time with a new state s_{k+1}
13: store transition (s_k, a_k, r_k, s_{k+1}) in D
14: sample random minibatch of transitions (s_k, a_k, r_k, s_{k+1}) from D
15: set $\text{target}_k = \begin{cases} r_k & \text{if episode terminates at step } k+1 \\ r_k + \gamma \max\limits_{a'} \widehat{Q}\left(\phi_{k+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
16: perform a gradient descent step on $(target_k - Q(s_k, a_k; \theta))^2$ with respect to the network parameters θ
17: **end**

In planning phase B (see Algorithm 4), we first build a resource pool that consists of all unused resources of cloud services (lines 2–5), and then blocked requests of all cloud services are transferred to the resource pool to execution (lines 6–8). We calculate the time from the current moment to the next planning stage A and the average number of task requests per minute to estimate the resources to be reserved for the next planning stage A, release a certain number of VMs (line 9).

Algorithm 4. Pseudo code for Planning Phase B

Input: The number of requests blocked $NumWait_i(\triangle t)$
Number of remaining available CPU resources $Remain_i(\triangle t)$
Average number of requests for cloud service $Cloud_i$ $AvgReq_i$
The time of counter k

Output: Task migration.

1: **begin**
2: set a resource pool
3: **for** (every cloud service $Cloud_i$ in SaaS provider at the interval Δt) **do**
4: $Remain_i(\triangle t)$ join the resource pool
5: **end for**
6: **for** (every cloud service $Cloud_i$ in SaaS provider at the interval Δt) **do**
7: Transfer The number of requests blocked $NumWait_i(\triangle t)$ to the resource pool
8: **end for**
9: **if** $k \equiv p$ mod 5 **then** reserve $(4 - p) \times AvgReq_i$ free resources to free the remaining empty VMs.
10: **end if**
11: **end**

(4) Execution Phase

In the execution phase A, the VM manager component executes the actions determined in the planning phase A. The VM manager creates a new VM for the cloud service $Cloud_i$ based on Table 2 or releases the lowest CPU utilization for scale-out action or scale-in action.

In the execution phase B, based on the determined behavior in the planning phase B, the resource manager performs task migration, and releases the VM according to the judgment criteria of the planning phase.

6 Evaluation

In this section, we present an experimental evaluation of the approach proposed in the previous section. First, we describe the simulation settings and performance indicators, and then introduce and discuss the experimental results.

6.1 Experimental Setup

We simulate 5 sets of cloud services to work together, and take 5 data segments with the same length from the load tracking of the Google data center [19] to evaluate our algorithm. The use of real load tracking makes the results more real and reliable, and can better reflect the randomness and volatility of user requests. The time interval is 5 min, the data segment lasts for more than 16 h, including 200 intervals. The selected workload data segments are shown in Fig. 4, corresponding to cloud service 0 cloud service 1 cloud service 2 cloud service 3 cloud service 4 respectively. In addition, we

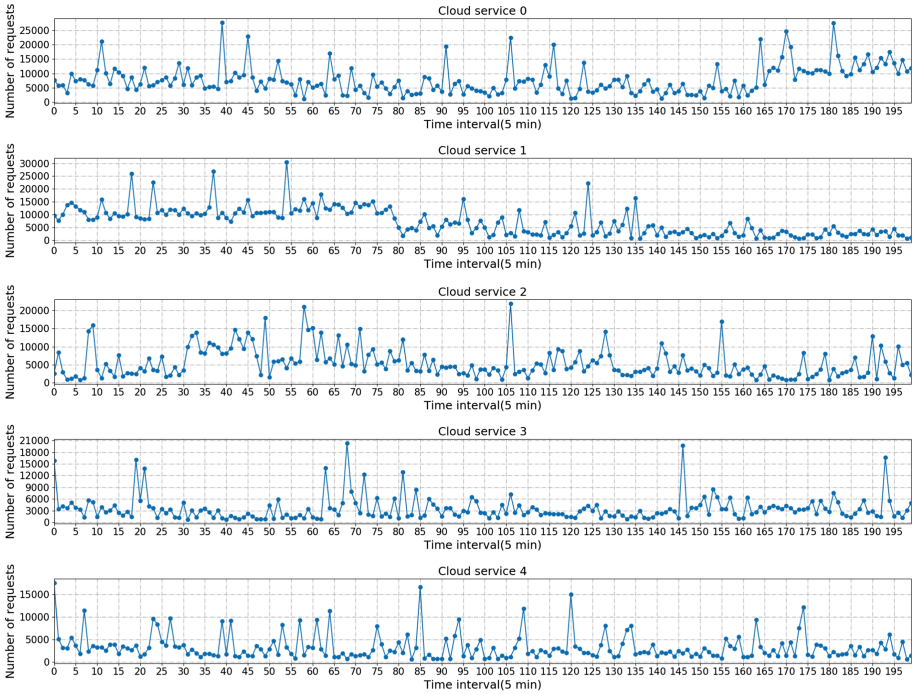

Fig. 4. Workload patterns for cloud service 0, cloud service 1, cloud service 2, cloud service 3 and cloud service 4.

Table 2. Different settings of VMs

VM type	Extra large	Large	Medium	Small
Core	8	4	2	1
CPU (MIPS)	20000	10000	5000	2500
RAM (GB)	150	75	37.5	17
Disk (GB)	16000	8500	4100	1600
VM price ($/h)	28	14	7	3.75
VM initiation price ($)	0.27	0.13	0.06	0.025

assume that the general scheduling strategy is first come first service strategy. There are four types of VMs offered by an IaaS provider: small, medium, large, and extra-large and they have different capacities and costs, as shown in Table 2.

For comparison, we conducted a group of experiments with our policy (Proposed). The contrast polices are as follow:

1. Constant resource policy (Constant): this policy is allocated with a static, constant number of cloud service resources.
2. Basic threshold-based policy (Threshold) [5]: this policy sets upper and lower bounds on the performance trigger adaptations to determines the horizontal scaling of cloud service resources.
3. RL controller-based policy (RL controller) [12]: this policy uses the workload prediction method and RL algorithm.

We applied the following metrics for a comparison of our policy with other strategies:

Utilization: The utilization of the cloud service $Cloud_i$ at the Δt^{th} interval is defined as the ratio of the resources occupied by user requests to the total CPU resources offered by VMs, at the Δt^{th} interval, and is defined as:

$$Utili_i(\Delta t) = \frac{Allocated(\Delta t)}{Total(\Delta t)} \tag{9}$$

Wasted Resources: The wasted resources of the cloud service $Cloud_i$ at the Δt^{th} interval is defined as the difference between the resources occupied by user requests and the total resources offered by VMs, at the Δt^{th} interval, and it can be described as:

$$Remain_i(\Delta t) = Allocated(\Delta t) - Total(\Delta t) \tag{10}$$

SLA Violation: The percentage SLA violation of the cloud service $Cloud_i$ at the Δt^{th} interval for user request Req_u^r of cloud service $Cloud_i$ at the Δt^{th} interval, which is calculated as:

$$SLAV_i(\Delta t) = \frac{1}{C}\sum_{c=1}^{C} SLA\left(Req_u^r\right) \tag{11}$$

Load Level: The load level of the cloud service $Cloud_i$ at the Δt^{th} interval is defined as ratio of the amount of resources required by the cloud service $Cloud_i$ to meet all requests to the amount of resources owned by the cloud service $Cloud_i$ at the moment, at the Δt^{th} interval, which is calculated as:

$$Load_i(\Delta t) = \frac{Need(\Delta t)}{Total(\Delta t)} \tag{12}$$

Total Cost: This metric is defined as the total cost of all cloud services $Cloud_i$ incurred by the SaaS provider at the Δt^{th} interval, and is expressed as:

$$Total\ Cost_i(\Delta t) = VM\ \ Cost_i(\Delta t) + Penalty\ Cost_i(\Delta t) \tag{13}$$

Profit: This metric is defined as the profit gained by the SaaS provider to serve all requests of cloud services, and is expressed as:

$$Profit_i(\Delta t) = Income_i(\Delta t) - Total\ Cost_i(\Delta t) \tag{14}$$

$$Income_i(\Delta t) = \sum_{u=1}^{U} \sum_{r=1}^{R} CPrice_i \times RTime_u^r \tag{15}$$

$CPrice_i$ is the service fee for the user request Req_u^r on the cloud service $Cloud_i$ and $RTime_u^k$ is the running time r^{th} request of the u^{th} user.

6.2 Experimental Results

Figure 5 shows the CPU utilizations of the four approaches for 5 workloads at each interval. The average CPU utilizations under 5 workloads for constant resource-based policy, basic threshold-based policy, RL controller-based policy, and the proposed policy are 54.56%, 85.17%, 78.14%, and 88.23%, respectively, as shown in Table 3. From the results, we observe that basic threshold-based policy, RL controller-based policy and

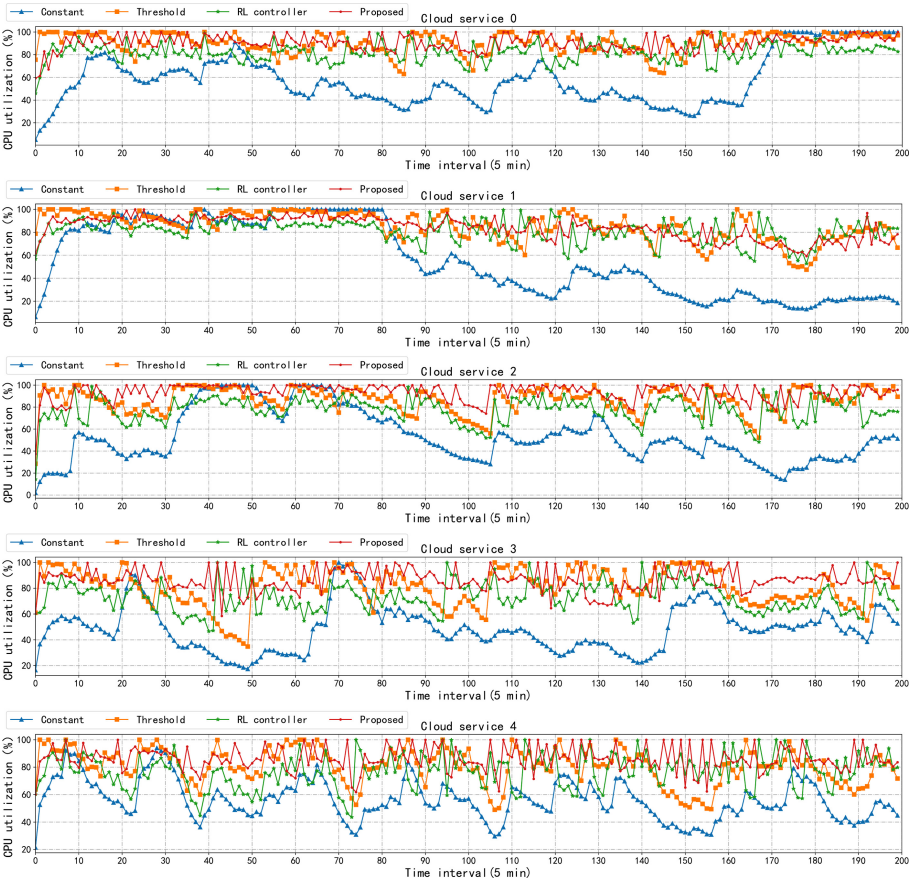

Fig. 5. Comparisons of CPU utilization of four polices for different cloud services.

Table 3. The average CPU utilization of four polices for different cloud services

	Constant	Threshold	RL controller	Proposed
Cloud service 0	58.62	**90.81**	81.83	90.68
Cloud service 1	56.07	**85.76**	80.95	84.23
Cloud service 2	53.34	86.87	77.49	**93.57**
Cloud service 3	48.57	81.41	73.26	**86.37**
Cloud service 4	56.20	80.99	77.18	**86.31**
Average	54.56	85.17	78.14	**88.23**

the proposed policy are able to utilize resources more fully, and the constant resource-based policy wastes more resources in 5 workloads for most intervals. This is because constant resource-based policy statically allocates a lower number of VMs. If the constant resource-based policy allocates enough VMs statically, the utilization will always be 100%, and there will be more waste of resources.

Figure 6 shows the waste of resources of the four polices for 5 workloads at each interval. The average resources remaining under 5 workloads for constant resource-based policy, basic threshold-based policy, RL controller-based policy, and the proposed policy are 30535.59, 5512.49, 9849.35 and 4315.68, respectively, as shown in Table 4. Our proposed policy outperforms other policies by reducing about 22%–86% of the resources wasting in 5 workloads.

The under-provisioning of resources causes SLA violations, and subsequently, leads to lower profit and fewer users. Compare with constant resource-based policy and basic threshold-based policy, RL controller-based policy has prediction-based controller, it is able to prepare sufficient resources in advance. Moreover, the pro-posed policy maintains a resource pool to make up for the prediction error of the prediction-based controller to reduce SLA violation rate. The proposed policy can reduce the SLA violations to less than 0.5%, as shown in Table 5.

Figure 7 shows the CPU load of the four polices for 5 workloads at each interval. The average CPU load under 5 workloads for constant resource-based policy, basic threshold-based policy, RL controller-based policy, and the proposed policy are 55.29%, 89.44%, 78.61%, and 88.27%, respectively, as shown in Table 6. The average value of the standard deviation (SD) about CPU load under 5 workloads for constant resource-based policy, basic threshold-based policy, RL controller-based policy, and the proposed policy are 23.31, 24.55, 11.63, and 8.46, respectively, as shown in Table 6. The smaller the standard deviation, the less volatile the CPU load. The smaller the data fluctuation, the better the solution to the multi-goal optimization problem of reducing SLA violation rate and increasing CPU utilization. The proposed policy gets a much higher standard deviation, so the proposed policy effectively reduced the SLA violation rate and increased the CPU utilization.

Our method is that multiple cloud services work together, we need to consider all cloud service costs and compare their sums. As mentioned above, the total cost depends on the cost of SLA violation and the cost of rented VMs. Our approach wastes the

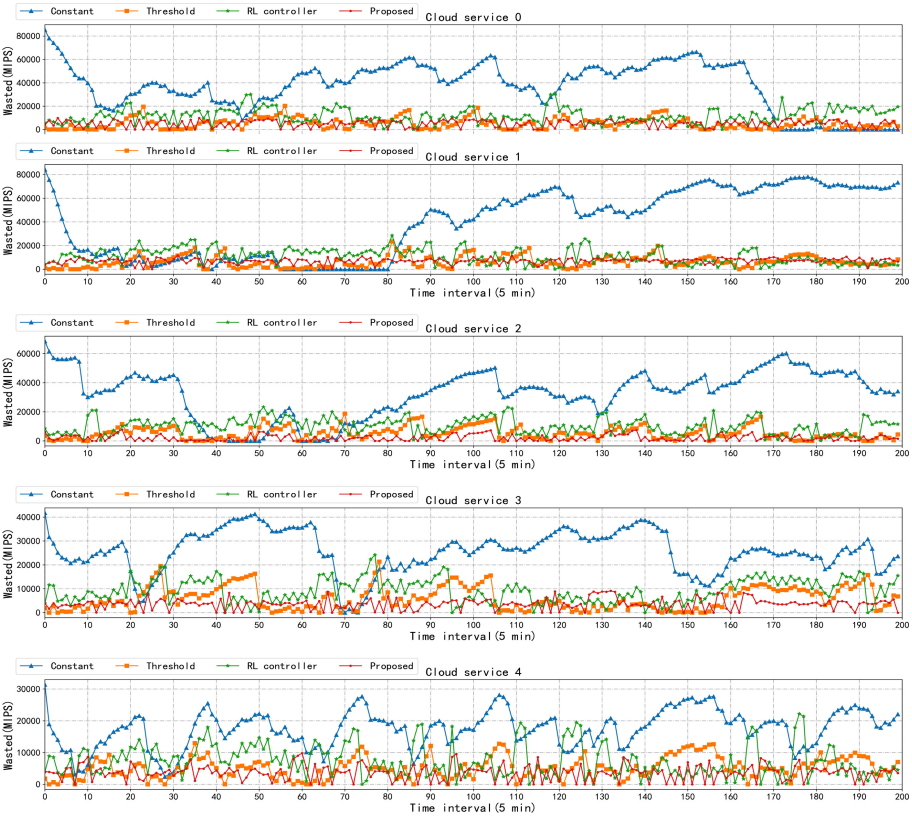

Fig. 6. Comparisons of wasted resources of four polices for different cloud services.

Table 4. The average wasted resources of four polices for different cloud services

	Constant	Threshold	RL controller	Proposed
Cloud service 0	37243.41	5103.99	11706.33	**5074.52**
Cloud service 1	39537.41	**6221.43**	10774.91	7122.11
Cloud service 2	32663.21	5221.75	10315.04	**2055.44**
Cloud service 3	25715.65	5872.27	9169.47	**3648.96**
Cloud service 4	17518.29	5143.03	7280.98	**3677.36**
Average	30535.59	5512.49	9849.35	**4315.68**

Table 5. The average SLA violations of four polices for different cloud services

	Constant	Threshold	RL controller	Proposed
Cloud service 0	13.30	23.38	3.26	**0.00**
Cloud service 1	14.16	14.49	2.72	**0.00**
Cloud service 2	7.57	16.11	1.66	**0.31**
Cloud service 3	1.01	13.96	3.61	**0.40**
Cloud service 4	**0.00**	11.55	6.65	0.30
Average	7.21	15.90	3.58	**0.20**

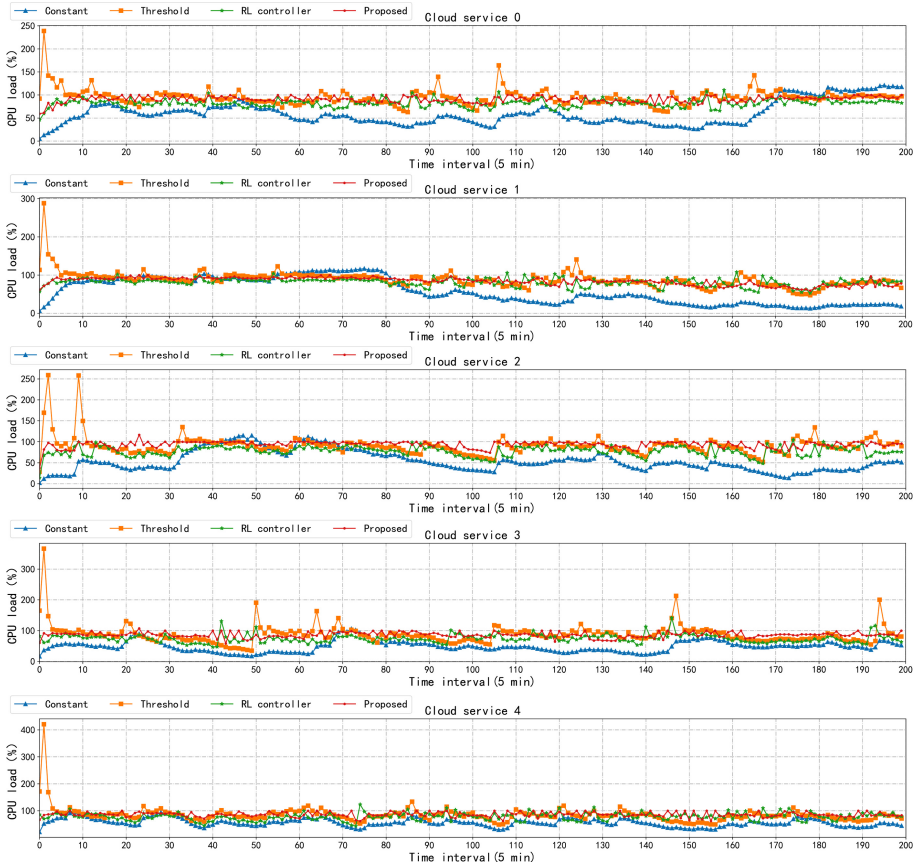

Fig. 7. Comparisons of CPU load of four polices for different cloud services.

Table 6. The average CPU load and the standard deviation of CPU load of four polices for different cloud services

	Constant (Ave/SD)	Threshold (Ave/SD)	RL controller (Ave/SD)	Proposed (Ave/SD)
Cloud service 0	60.23/25.77	**94.89**/17.50	82.12/8.77	90.68/**6.94**
Cloud service 1	57.45/33.67	**88.84**/21.20	81.15/10.33	84.23/**9.64**
Cloud service 2	53.92/24.88	90.91/23.04	77.63/11.81	**93.65/8.00**
Cloud service 3	48.62/17.81	**87.59**/31.41	74.12/13.40	86.41/**8.89**
Cloud service 4	56.20/14.43	84.97/29.65	78.04/13.88	**86.35/8.86**
Average	55.29/23.31	**89.44**/24.56	78.61/11.64	88.27/**8.46**

least amount of resources and has the lowest rate of violations, as shown in Table 4 and Table 5. Figure 8 shows the total cost of the four approaches for 5 workloads at all interval. The proposed policy outperforms other approaches by saving cost about 17%–58% in 5 workloads, as shown in Table 7.

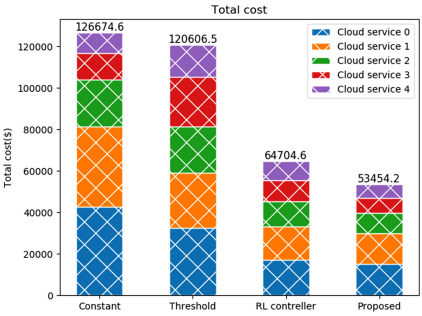

Fig. 8. The sum of total cost at five cloud service for different polices.

Table 7. The total cost of four polices for different cloud services.

	Constant	Threshold	RL controller	Proposed
Cloud service 0	42591.21	32392.45	17124.12	**15111.93**
Cloud service 1	38700.50	26726.88	16034.35	**14783.79**
Cloud service 2	22593.87	22145.12	12178.67	**9819.02**
Cloud service 3	12788.99	23848.74	10057.89	**7204.37**
Cloud service 4	10000.00	15493.29	9309.60	**6535.13**
Sum	126674.57	120606.47	64704.62	**53454.25**

Figure 9 shows the profit of the four approaches for 5 workloads at all interval, and we observe that basic RL controller-based policy and the proposed policy are able to make considerable profits, as shown in Table 8. Table 5 shows the proposed policy less on SLA violations than what RL controller-based policy and guarantee the quality of cloud service to increased customer retention. Therefore, the proposed policy can make the most profit in the long term.

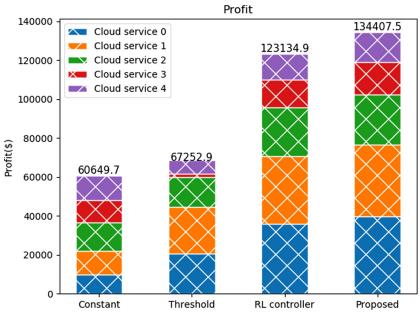

Fig. 9. The sum of profit at five cloud service for different polices.

Table 8. The profit of four polices for different cloud services.

	Constant	Threshold	RL controller	Proposed
Cloud service 0	9978.23	20750.97	35982.77	**39780.51**
Cloud service 1	11931.88	23939.15	34598.04	**36656.28**
Cloud service 2	14750.14	15169.65	25162.92	**25904.66**
Cloud service 3	11484.88	376.59	14196.51	**16590.64**
Cloud service 4	12504.54	7016.54	13194.70	**15475.38**
Sum	60649.68	67252.91	123134.94	**134407.46**

Based on the above results, we compared the CPU utilization, resources remaining, SLA violations, CPU load, total cost and profit of four approaches under 5 workloads. We observed that the proposed approach increases the CPU utilization by up to 3%–34%, the profit by up to not less than 9% and reduces the waste of resources by up to about 22%–86%, the SLA violations to less than 0.4%, the total cost saving about 17%–58% compared with the other polices.

7 Conclusion

In this study, we proposed an elastic resource scheduling algorithm based on a combination of the autonomic computing and DRL. In order to implemented the proposed approach, we presented a resource provisioning framework that supported the MAPE control loop, and used a two-tier elastic resource configuration. Our framework could adjust the number of VMs in advance by workload prediction and DRL algorithm to achieve horizontal scaling, and achieve vertical scaling by resource sharing, collaborative work and self-adapting management. Experimental results using actual Google cluster tracking results showed that the proposed approach could increase the resource utilization and decrease the total cost, while avoiding SLA violations. In the future, we plan to consider the type of request based on the proposed method to configure a more suitable cloud service. In addition, some emergency situations such as the sharp fluctuations of user requests and VM failures will be considered.

Acknowledgements. The authors are grateful for the support of the National Natural Science Foundation of China (61373149), the Natural Science Foundation of Shandong Province (ZR2020QF026) and the project of Shandong Normal University (2018Z29).

References

1. Singh, S., Chana, I.: Cloud resource provisioning: survey, status and future research directions. Knowl. Inf. Syst. **49**(3), 1005–1069 (2016)
2. Yousafzai, A., Gani, A., Noor, R.M., et al.: Cloud resource allocation schemes: review, taxonomy, and opportunities. Knowl. Inf. Syst. **50**(2), 347–381 (2017)
3. Suresh, A., Varatharajan, R.: Competent resource provisioning and distribution techniques for cloud computing environment. Cluster Comput. 1–8 (2019)
4. Chieu, T.C., Mohindra, A., Karve, A.A., et al.: Dynamic scaling of web applications in a virtualized cloud computing environment. In: 2009 IEEE International Conference on e-Business Engineering, pp. 281–286. IEEE (2009)
5. Yang, J., Liu, C., Shang, Y., et al.: A cost-aware auto-scaling approach using the workload prediction in service clouds. Inf. Syst. Front. **16**(1), 7–18 (2014)
6. Roy, N., Dubey, A., Gokhale, A.: Efficient autoscaling in the cloud using predictive models for workload forecasting. In: 2011 IEEE 4th International Conference on Cloud Computing, pp. 500–507. IEEE (2011)
7. Weingärtner, R., Bräscher, G.B., Westphall, C.B.: Cloud resource management: a survey on forecasting and profiling models. J. Netw. Comput. Appl. **47**, 99–106 (2015)
8. Nikravesh, A.Y., Ajila, S.A., Lung, C.H.: An autonomic prediction suite for cloud resource provisioning. J. Cloud Comput. **6**(1), 3 (2017)
9. Mazidi, A., Golsorkhtabaramiri, M., Tabari, M.Y.: Autonomic resource provisioning for multi-layer cloud applications with K-nearest neighbor resource scaling and priority-based resource allocation. Softw. Pract. Exp. (2020)
10. Wei, Y., Kudenko, D., Liu, S., et al.: A reinforcement learning based auto-scaling approach for SaaS providers in dynamic cloud environment. Math. Probl. Eng. **2019** (2019)
11. Ghobaei-Arani, M., Jabbehdari, S., Pourmina, M.A.: An autonomic resource provisioning approach for service-based cloud applications: a hybrid approach. Future Gener. Comput. Syst. **78**, 191–210 (2018)

12. Li, Q., Hao, Q., Xiao, L., et al.: An integrated approach to automatic management of virtualized resources in cloud environments. Comput. J. **54**(6), 905–919 (2011)
13. Gill, S.S., Buyya, R., Chana, I., et al.: BULLET: particle swarm optimization based scheduling technique for provisioned cloud resources. J. Netw. Syst. Manag. **26**(2), 361–400 (2018)
14. Salah, K., Elbadawi, K., Boutaba, R.: An analytical model for estimating cloud resources of elastic services. J. Netw. Syst. Manag. **24**(2), 285–308 (2016)
15. Wu, L., Garg, S.K., Buyya, R.: SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments. In: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 195–204. IEEE (2011)
16. Huebscher, M.C., Mccann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. **40**(3), 1–28 (2008)
17. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
18. Sowell, F.: Modeling long-run behavior with the fractional ARIMA model. J. Monet. Econ. **29**(2), 277–302 (1992)
19. Google Cluster-Usage Traces. http://code.google.com/p/googleclusterdata