Resource Allocation With Workload-Time Windows for Cloud-Based Software Services: A Deep Reinforcement Learning Approach

Xing Chen, *Member, IEEE*, Lijian Yang, Zheyi Chen, Geyong Min, *Member, IEEE*, Xianghan Zheng, and Chunming Rong, *Senior Member, IEEE*

Abstract—As the workloads and service requests in cloud computing environments change constantly, cloud-based software services need to adaptively allocate resources for ensuring the Quality-of-Service (QoS) while reducing resource costs. However, it is very challenging to achieve adaptive resource allocation for cloud-based software services with complex and variable system states. Most of the existing methods only consider the current condition of workloads, and thus cannot well adapt to real-world cloud environments subject to fluctuating workloads. To address this challenge, we propose a novel Deep Reinforcement learning based resource Allocation method with workload-time Windows (DRAW) for cloud-based software services that considers both the current and future workloads in the resource allocation process. Specifically, an original Deep Q-Network (DQN) based prediction model of management operations is trained based on workload-time windows, which can be used to predict appropriate management operations under different system states. Next, a new feedback-control mechanism is designed to construct the objective resource allocation plan under the current system state through iterative execution of management operations. Extensive simulation results demonstrate that the prediction accuracy of management operations generated by the proposed DRAW method can reach 90.69%. Moreover, the DRAW can achieve the optimal/near-optimal performance and outperform other classic methods by 3$\sim$13% under different scenarios.

Index Terms—Cloud-based software services, resource allocation, workload-time windows, deep reinforcement learning, feedback control

1 Introduction

Cloud datacenters consist of various resources, such as central processing units (CPUs), memories, and storage units. Through utilizing virtualization technologies, these cloud resources can be used to offer rental services [1]. Commonly, it is expected that the available cloud resources should be provided to the authorized users for accessing purposes according to their service requests. However, in cloud environments, workloads and service requests are changing constantly, which may severely downgrade the Quality-of-Service (QoS) and cause excessive resource costs [2]. To ensure the scalability and resiliency of resources, cloud service providers (CSPs) should be able to allow on-demand configurations for software and hardware resources in shared cloud infrastructures [3].

Over the years, extensive applications of cloud-based software services have witnessed their rapid growth [4]. However, due to complex and variable system states in cloud environments, it is very challenging to achieve a good trade-off between the QoS and resource costs when making resource allocation for cloud-based software services [5]. To address this challenge, it is crucial to design an adaptive resource allocation method for cloud-based software services. There are some classic methods, such as rule-based [6], heuristic [7], control-theoretic [8], and machine learning (ML) based [9], and reinforcement learning (RL) based [10] methods, that can address the problem of cloud resource allocation to some extent. The rule-based methods need to define various rules for a software service, which results in huge costs of rule settings and limits their applications in dynamic cloud environments. The heuristic methods may lead to excessive searching costs when the solution space is large, and meanwhile the deviation degree of the feasible plans from the optimal one is unpredictable. The control-theoretic methods need a large number of iterations to determine an effective resource allocation plan, which may lead to additional costs if virtual machines (VMs) are frequently discontinued. The ML-based methods enable cloud systems with an ability to learn the specific domain knowledge from historical data for optimizing resource allocation. However, in real-world cloud environments, it is hard to collect enough training data to support the ML-based

- *Xing Chen, Lijian Yang, Zheyi Chen, and Xianghan Zheng are with the College of Computer and Data Science, Fuzhou University, Fuzhou 350116, China and also with the Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou 350116, China. E-mail: {Chenxing, z.chen, xianghan.zheng}@fzu.edu.cn, yangljian@foxmail.com.*
- *Geyong Min is with the Department of Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, EX4 4QF Exeter, U.K. E-mail: G.Min@exeter.ac.uk.*
- *Chunming Rong is with the Department of Electronic Engineering and Computer Science, University of Stavanger, 4036 Stavanger, Norway. E-mail: chunming.rong@uis.no.*

Manuscript received 24 Nov. 2021; revised 16 Apr. 2022; accepted 18 Apr. 2022. Date of publication 21 Apr. 2022; date of current version 7 June 2023.
This work was supported in part by the National Natural Science Foundation of China under Grant 62072108 and in part by the Natural Science Foundation of Fujian Province for Distinguished Young Scholars under Grant 2020J06014.
(Corresponding authors: Zheyi Chen and Geyong Min.)
Recommended for acceptance by C. Wu.
Digital Object Identifier no. 10.1109/TCC.2022.3169157

methods. The RL-based methods to retrain their decision-making models if workload changes happen because they target at the environment with static workloads. Therefore, these classic methods of resource allocation for cloud-based software services might not well fit in real-world cloud environments with changeable workloads and service requests.

In the problem of resource allocation for cloud-based software services, the condition of workloads has been proven to be an important determinant that sways the QoS and resource costs [11]. When numerous service requests arrive within a short time, the workloads on cloud datacenters would burst and the existing resources may not well meet the demands of cloud-based software services. However, when the workloads stay low with few service requests, unnecessary resource allocation might result in great waste. For this purpose, workload prediction has become a promising solution to improve the effect of resource allocation in cloud environments [12]. Commonly, the effectiveness of resource allocation would be declined when making decisions of resource allocation based on the current workload only, without considering future workload changes. However, it would be hard to achieve an accurate prediction for future workloads, which may seriously degrade the performance of resource allocation in the scheduling stage [13]. Some research works, such as [14] and [15], to cite two of the most recent, promised high prediction accuracy for future workloads, which contributed to improving the scheduling efficiency. Built upon these existing work, it is necessary to introduce the workload-time windows for further supporting the performance improvement of resource allocation.

Therefore, we propose an original Deep Reinforcement learning based resource Allocation method with workload-time Windows (DRAW) for cloud-based software services. The main contributions of this paper are summarized as follows.

- Through introducing workload-time windows, we define the resource allocation for cloud-based software services as an optimization problem with the consideration of both the current and future workloads. The optimization objective of this resource allocation problem is to make a good trade-off between the QoS and resource costs, where a fitness function is well designed to measure the effectiveness of resource allocation plans.
- A novel Deep Reinforcement learning based resource Allocation method with workload-time Windows (DRAW) is proposed for cloud-based software services. First, we train a DQN-based prediction model of management operations based on workload-time windows, which can be used to predict appropriate management operations under different system states. Next, we design a feedback-control mechanism to explore the objective resource allocation plan under the current system state through iterative execution of management operations.
- Using the RUBiS benchmark, extensive simulation experiments are carried out to verify the effectiveness of the proposed DRAW for cloud-based software services. The performance results show that the prediction accuracy of management operations can reach 90.69% by the proposed DRAW. Moreover, the DRAW is able to achieve the optimal/near-optimal performance in terms of the QoS and resource costs, and it exceeds classic methods by 3$\sim$13% under various scenarios.

The rest of this paper is organized as follows. Section 2 reviews the related work. In Section 3, the problem of resource allocation is defined. Section 4 presents the proposed DRAW in detail. In Section 5, the evaluation results are analyzed. Section 6 concludes this paper.

2 Related Work

The problem of cloud resource allocation has attracted much attention in the research community, and many scholars have contributed to solving this important problem. In this section, we review and analyze the related work by classifying the existing methods.

The rule-based methods use various rule settings to satisfy service demands. For example, Farooq *et al.* [6] adopted a pricing policy based on the Quality-of-Experience (QoE) of applications and made resource allocation rules by using the statistical information of the current workload. Abdullah *et al.* [16] designed a rule-based autoscaling method for IoT microservices in the fog based on the predicted workload, in order to reduce the response time. Buchaca *et al.* [17] proposed a vertical auto-scaling strategy to resize containers according to the results of workload prediction. However, massive rules need to be defined for different software services when using the rule-based methods, which not only limits their applications in dynamic cloud environments but also caused huge costs of rule settings.

Heuristic methods use individual or global experience to formulate and update searching strategies of resource allocation plans. Ardagna *et al.* [7] developed a joint strategy of load balancing and receding horizon capacity allocation based on the workload prediction, in order to minimize execution overheads while satisfying the delay constraints of cloud-based applications. Khatua *et al.* [18] utilized some heuristic-based polynomial time algorithms to explore the near-optimal solution of resource reservation with the consideration of demand forecasting. Chen *et al.* [19] introduced future workloads and proposed a particle swarm optimization and genetic algorithm (PSO-GA) based resource allocation method for cloud-based software services. However, when the solution space is large, it may cause excessive computational costs by heuristic searching, and the deviation degree of the feasible plans from the optimal one is unpredictable.

The control-theoretic methods commonly use feedback loops to make resource allocation decisions for cloud services. Haratian *et al.* [8] designed a resource management framework to satisfy QoS demands, where a fuzzy controller was used to determine the resource allocation based on the current workload changes during each iteration. Tolosana-Calasanz *et al.* [20] proposed an autonomic controller based on the feedback control and queueing theory, which can elastically allocate VM resources to data streams with periodically-fixed workloads for satisfying their performance goals. Hoseiny-Farahabady *et al.* [21] used close-loop control theory to design a resource adjustment mechanism for satisfying various QoS requirements of applications, where the incoming workload intensity is predicted for the next controlling intervals. Makridis *et al.* [22] developed two filter-based robust controllers to allocate application resources, which can offer improved performance under abrupt workload changes within a time window. However, a large number of iterations are needed to determine an effective resource allocation plan when using the control-theoretic methods, which may lead to additional costs if VMs are frequently discontinued.

The ML-based methods enable cloud systems with an ability to learn the specific domain knowledge from historical data for optimizing resource allocation. Wei *et al.* [9] designed a skewness-avoidance multi-resource (SAMR) allocation method to allocate heterogeneous workloads into physical machines, where the workloads are predicted as each time slot begins. Chen *et al.* [23] proposed an ML-based resource allocation method for cloud-based software services, where the QoS prediction model was constructed based on the current workload. Wang *et al.* [24] extracted the similarity among different cloud environments by using an ML-based method, and thus the optimal/near-optimal resource allocation plan can be found for the current scenario. Choochotkaew *et al.* [25] first designed a regression-based method to predict resource requests in the next window and then applied ML techniques to determine whether the allocated resources can meet response demands according to predicted workloads. However, in real-world cloud environments, it is hard to collect enough training data to support the ML-based methods, which may severely harm the model accuracy of resource allocation.

Reinforcement learning (RL) [32] makes decisions of resource allocation via interacting with an environment without inputting historical data. Liu *et al.* [10] proposed a deep RL-based hierarchical framework to handle the problems of resource allocation and power management in the cloud. Based on an advantage actor-critic RL method, Chen *et al.* [26] optimized the resource allocation problem by reducing the system delay when scheduling jobs. Alsarhan *et al.* [27] designed a service level agreement (SLA) framework by integrating RL to make VM hiring policies, which can well adapt to dynamic cloud environments and satisfy user demands. Orhean *et al.* [28] adopted the Q-learning algorithm to schedule the heterogeneous nodes with dependent tasks in distributed systems for reducing the execution time. Chen *et al.* [29] utilized the Q-learning algorithm to evaluate the values of management operations, in order to better handle the resource allocation problem of cloud-based software services. Guo *et al.* [30] developed a deep RL-based strategy to address the problem of cloud resource management, where the imitation learning is used to reduce the RL training time of exploring the optimal policy. Wang *et al.* [31] presented a VM scheduling mechanism to balance the QoS and energy consumption based on the Q-learning algorithm and QoS feature extraction. However, these RL-based approaches aim at the environment with static workloads, and it is necessary to retrain their decision-making models if workload changes happen. Moreover, these approaches do not well consider future workload changes. When making decisions of resource allocation based on the current workload only, the effectiveness of resource allocation would be declined.

In general, these classic methods cannot well adapt to the resource allocation problem for cloud-based software services in real-world cloud environments with variable workloads and service requests. To better handle this problem, it is necessary to design a novel resource allocation method with the consideration of workload-time windows.

3 Problem Definition

The changing runtime environments can lead to different levels of the QoS in cloud-based software services. Specifically, the external changes of runtime environments are determined by external factors (e.g., dynamic workloads), while the internal ones are influenced by internal factors in cloud management systems (e.g., different types, numbers, and rental prices of VMs in resource allocation plans).

Assume that a continuous function is used to describe the workload changes, which is defined as

$$W = \{w(t) \mid t_0 \leq t \leq t_0 + T\}, \tag{1}$$

where the workload at the time t_i is $w(t_i)$, consisting of the workload amount and types. The workload amount and types represent the total number of service requests and the proportions of various service requests. $t_0 \leq t \leq t_0 + T$ represents the time period for which resources can be adjusted.

Moreover, a resource allocation plan is defined as

$$VM = \{vm(t) \mid t_0 \leq t \leq t_0 + T\}. \tag{2}$$

Considering that there are n types of VMs in a cloud system, and thus the resource allocation plan corresponding to the workload at t_i is defined as

$$vm(t_i) = \left\{vm^1_{t_i}, vm^2_{t_i}, \ldots vm^n_{t_i}\right\}, \tag{3}$$

where $vm^j_{t_i}$ indicates the number of VMs of the jth type.

Commonly, the objective of resource allocation for cloud-based software services is to enhance the QoS and reduce resource costs while making a good trade-off between them, which can be defined by an objective function.

On one hand, QoS_i represents the QoS at t_i, which follows the SLAs with two performance metrics, including the response time (RT_i) and the data throughput (DH_i) [33]. Thus, QoS_i is defined as

$$QoS_i = SLAs(RT_i, DH_i). \tag{4}$$

On the other hand, $Cost_i$ represents the resource costs at t_i, which includes the rental costs of VMs ($Cost_L$) and the costs of switching VMs ($Cost_D$). Assuming that there are n types of VMs, the total resource costs can be defined as

TABLE 1
Contrastive Analysis of Different Work ("+": Involved; "-": Not Involved)

Related work	Method					Workload	
	Rule-based	Heuristic	Control-theoretic	ML-based	RL-based	Current	Time window
Farooq *et al.* [6]	+	-	-	-	-	+	-
Abdullah *et al.* [16]	+	-	-	-	-	-	+
Buchaca *et al.* [17]	+	-	-	-	-	-	+
Ardagna *et al.* [7]	-	+	-	-	-	+	-
Khatua *et al.* [18]	-	+	-	-	-	+	-
Chen *et al.* [19]	-	+	-	-	-	-	+
Haratian *et al.* [8]	-	-	+	-	-	+	-
Tolosana-Calasanz *et al.* [20]	-	-	+	-	-	+	-
HoseinyFarahabady *et al.* [21]	-	-	+	-	-	-	+
Makridis *et al.* [22]	-	-	+	-	-	-	+
Wei *et al.* [9]	-	-	-	+	-	-	+
Chen *et al.* [23]	-	-	-	+	-	+	-
Wang *et al.* [24]	-	-	-	+	-	+	-
Choochotkaew *et al.* [25]	-	-	-	+	-	-	+
Liu *et al.* [10]	-	-	-	-	+	+	-
Chen *et al.* [26]	-	-	-	-	+	+	-
Alsarhan *et al.* [27]	-	-	-	-	+	+	-
Orhean *et al.* [28]	-	-	-	-	+	+	-
Chen *et al.* [29]	-	-	-	-	+	+	-
Guo *et al.* [30]	-	-	-	-	+	+	-
Wang *et al.* [31]	-	-	-	-	+	+	-
Proposed DRAW	-	-	-	-	+	-	+

$$\int_{t_0}^{t_0+T} Cost\ \mathrm{dt} = \int_{t_0}^{t_0+T} Cost_L\ \mathrm{dt} + Cost_D = \sum_{j=1}^{n}\int_{t_0}^{t_0+T}\left(vm_t^j \times C_j\right)\mathrm{dt} + \sum_{j=1}^{n} S_j \times D_j, \quad (5)$$

where the unit cost of renting the jth type of VMs is C_j, the unit cost of switching the jth type of VMs is D_j, and S_j is the number of VMs of the jth type that are switched during the resource adjustment time T.

Therefore, the objective function is defined as

$$\text{Minimize } r_1 \times \int_{t_0}^{t_0+T} \frac{1}{QoS}\ \mathrm{dt} + r_2 \times \int_{t_0}^{t_0+T} Cost\ \mathrm{dt}, \quad (6)$$

where r_1 and r_2, predefined by cloud engineers, are used to weight the QoS and resource costs, respectively.

However, the effectiveness of resource allocation cannot be well insured when making resource allocation plans based on the current workload only, without considering future workload changes. By introducing workload-time windows, the future workloads can be used to better support the decision-making process of resource allocation at the current time. Accordingly, the current resource allocation plan can be well adjusted and thus become more effective. Fig. 1 illustrates the definition of a workload-time window.

Assuming that the time length of predictable workloads is L ($L \ll T$), and the workload at t_i in workload-time windows can be expressed as

$$W^L = \{w(t) \mid t_i \leq t \leq t_i + L\}. \quad (7)$$

Accordingly, the optional resource allocation plan is defined as

$$VM_{opt}^L = \{VM_{t_i}, VM_{t_i+\Delta p}, \ldots, VM_{t_i+(l-1)\times\Delta p}\}_{opt}, \quad (8)$$

where Δp is the time interval for adjusting the resource allocation plan and $l = \frac{L}{\Delta p}$ is the number of times that the resource allocation plan needs to be adjusted in a workload-time window.

Therefore, the fitness function of the resource allocation plan at t_i in a workload-time window is defined as

$$Fitness^L = r_1 \times \int_{t_i}^{t_i+L} \frac{1}{QoS}\ \mathrm{dt} + r_2 \times \int_{t_i}^{t_i+L} Cost\ \mathrm{dt}. \quad (9)$$

In each workload-time window corresponding to different times, the objective resource allocation plan is with the smallest value of the fitness function.

Based on the above definitions, the runtime datasets in workload-time windows are shown in Table 2. The runtime state is defined as a 2-tuple, denoted by $\langle W^L, vm_{cur} \rangle$. For each state, there are multiple optional resource allocation plans in a workload-time window. Meanwhile, the fitness function of each optional resource allocation plan in VM_{opt}^L is defined as $fitness_{t_i}$. Therefore, the plan that minimizes $Fitness_{opt}^L$ in the current workload-time window is regarded as the objective resource allocation plan, denoted by VM_{obj}^L. Next, according to VM_{obj}^L, VM_{t_i} can be obtained and well adjusted.

However, it would be hard to obtain the objective resource allocation plan in complex cloud environments with changeable workloads and service requests. To address this problem, with the consideration of workload-time windows, deep reinforcement learning (DRL) will be used to establish a decision-making model of management

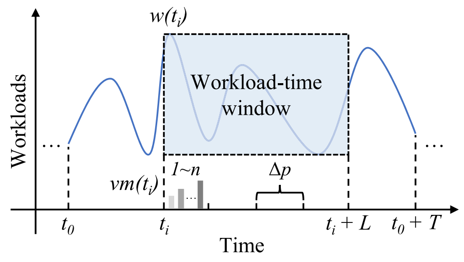

Fig. 1. Definition of a workload-time window.

operations, aiming to realize the adaptive resource allocation for cloud-based software services.

4 DRL-Based Resource Allocation With Workload-Time Windows

This section introduces the proposed Deep Reinforcement learning based resource Allocation method with workload-time Windows (DRAW) for cloud-based software services in detail. Fig. 2 shows an overview of the DRAW, where the main steps are given as follows.

Step 1: The training datasets of the DQN model are constructed based on historical running data. As shown in Table 3, the datasets contain the workload-time windows W^L, the current resource allocation plan VM_{cur} with the corresponding QoS value QoS_{cur}, and the objective resource allocation plan VM_{obj} (obtained by combining management experience and small-scale verification). Based on the above training datasets, the DQN algorithm [34] is used to train a Q-value prediction model of management operations, which can evaluate the Q-values of management operations under different system states.

Step 2: According to the current workload-time windows and resource allocation plan with corresponding QoS values, the Q-value prediction model is first used to predict the Q-values of various management operations. Next, the appropriate management operations will be selected based on their Q-values. Finally, the objective resource allocation plan can be obtained by feedback-control iterations.

4.1 DQN-Based Prediction of Management Operations

As an RL algorithm, the DQN is used to evaluate the Q-values of various management operations to enhance the effectiveness of cloud resource allocation. The DQN combines the Q-learning [32] with deep learning (DL) technologies [35], which replaces the Q-value table in the Q-learning with deep neural networks (DNNs). Therefore, there is no need for the DQN to make searches in a huge Q-value table because the DQN fits the Q-value function into DNNs. Through inputting a state, the Q-values of all corresponding actions can be obtained.

As shown in Fig. 3, the DQN agent first receives the state s in the runtime cloud environment and chooses the action a through the greedy strategy. Next, it obtains the corresponding reward r and steps to the next state s'. At each step, (s, a, r, s') is stored in the replay buffer with the predefined capacity. If the capacity threshold is reached, the parameters ω of DNNs will be updated, and the loss function is defined as

$$Loss = (r + \gamma \max Q(s', a'; \omega') - Q(s, a; \omega))^{2}, \tag{10}$$

where γ is the discount factor, ω is the weights of DNNs, $Q(s, a; \omega)$ is the current Q-value, and $\max Q(s', a'; \omega')$ is the maximum Q-value when choosing a' at s'.

For each runtime state in Table 2, VM_{obj} is explored and recorded in Table 3. Assuming there are d time points in a workload-time window, where $W^L = w(t_i)$ $(1 \le i \le d)$ is the workload at t_i, $VM_{cur} = (vm^1_{t_i}, vm^2_{t_i}, \ldots, vm^n_{t_i})$ is the resource allocation plan at t_i, $vm^j_{t_i}$ $(1 \le j \le n)$ is the number of the jth type of VMs at t_i, the QoS is QoS_{t_i} at t_i, and $VM^{obj}_{t_i}$ is the objective resource allocation plan at t_i.

Based on the training datasets, the DQN agent is able to find the management operations with the corresponding highest Q-values at varying runtime states. Commonly, the DRL problem can be formulated as the Markov decision process (MDP), which aims to maximize the long-term rewards. Specifically, an MDP can be described as a 4-tuple $\langle S, A, P, R\rangle$, where S, A, P, and R represent the state space, action space, state transition function, and reward function, respectively. According to the problem definition of resource allocation, they are defined as follows.

State space: S is defined as a 3-tuple $\langle W^L, VM_{cur}, QoS_{cur}\rangle$ described in Table 3, which indicates the current state of the runtime cloud environment.

Action space: $A = \{add_1, rm_1, add_2, rm_2, \ldots, add_m, rm_m\}$, where an action a $(a \in A)$ (i.e., management operation) adds or removes a VM of a type. For instance, add_i adds a VM of the ith type, and rm_i removes a VM of the ith type.

Reward function: R guides the DQN agent to explore VM_{obj}, which is defined as

$$R(VM_{cur}, VM_{obj}, a) = \begin{cases} 10, & getNext(VM_{cur}, a) == VM_{obj} \\ -1, & getNext(VM_{cur}, a) \notin VM_{opt} \\ 0, & getNext(VM_{cur}, a) \in VM_{opt} \end{cases}, \tag{11}$$

TABLE 2
Datasets of Runtime State and Optional Resource Allocation Plans With Fitness Function Towards Workload-Time Windows

Runtime state		Optional resource allocation plans with fitness function	
W^L	VM_{cur}	VM^L_{opt}	$Fitness^L_{opt}$
$w(t_1)$	$vm^1_{t_1}, vm^2_{t_1}, \ldots, vm^n_{t_1}$	$VM_{t_1}, VM_{t_1+\Delta p}, \ldots, VM_{t_1+l\times\Delta p}$	$fitness_{t_1}$
$w(t_2)$	$vm^1_{t_2}, vm^2_{t_2}, \ldots, vm^n_{t_2}$	$VM_{t_2}, VM_{t_2+\Delta p}, \ldots, VM_{t_2+l\times\Delta p}$	$fitness_{t_2}$
...	...	...	...
$w(t_u)$	$vm^1_{t_u}, vm^2_{t_u}, \ldots, vm^n_{t_u}$	$VM_{t_u}, VM_{t_u+\Delta p}, \ldots, VM_{t_u+l\times\Delta p}$	$fitness_{t_u}$

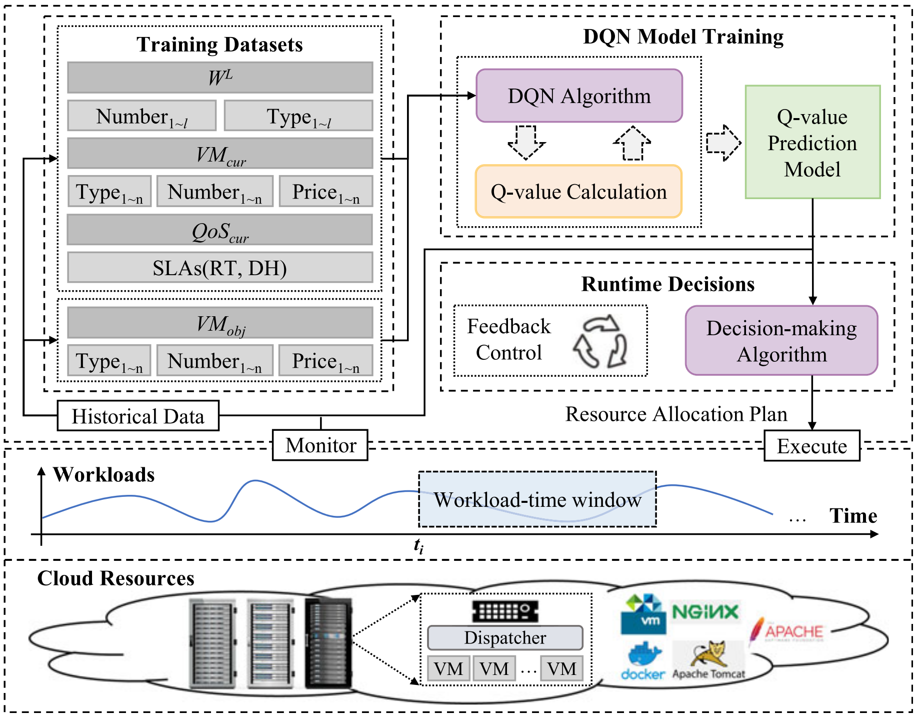

Fig. 2. Overview of the proposed DRAW.

where VM_{cur} is the current resource allocation plan, VM_{opt} is the optional resource allocation plan, and $getNext(VM_{cur}, a)$ indicates the next configuration of VMs after executing an action a under the current resource allocation plan VM_{cur}. If VM_{obj} can be found by taking an action a under VM_{cur}, $R = 10$. If a newly-generated resource allocation plan does not belong to VM_{opt} after taking an action a under VM_{cur}, $R = -1$. For other cases, $R = 0$ when taking actions.

The DQN determines the new Q-values according to the Q-values estimated by DNNs and the backpropagation of rewards, and it brings the Q-values into the loss function to update the parameters of DNNs. However, it is tough to directly train a Q-value prediction model by the above process. Taking Table 4 a as an example to explain this problem, where there are three types of VMs. Specifically, this table shows the Q-values of different management operations (i.e., add_2, add_3, rm_2, and rm_2) under various configurations of VMs, where the number of VMs of the 1st type is 0, the number of VMs of the 2nd type is changing from 2 to 8, and the number of VMs of the 3 rd type is changing from 2 to 6. The target state (i.e., the objective resource allocation plan) is underlined for each of these four divided areas. For example, when the current VM configuration is (0,2,2), the objective VM configuration is (0,4,4) in the corresponding area. For another example, when the current VM configuration is (0,5,2), the objective VM configuration is (0,5,4) in the corresponding area. Moreover, some special issues (highlighted) are further analyzed as follows.

- When staying in the target state, the theoretical Q-values of all management operations is 0. However, when approaching the target state, these Q-values are too high to achieve the accurate Q-value prediction of management operations. For example, when the numbers of VMs of different types are configured as (0,4,4), the predicted Q-value of add_2 is 2.9373 (the theoretical value should be 0), which will affect the setting of termination conditions in the runtime decision-making process. As shown in Table 4, when staying in the target state, the Q-values of management operations are close to 3. Therefore, when all Q-values are less than 3, the target state is regarded to be reached and the program will be terminated. It should be noticed that, when VMs are configured as (0,2,2), (0,2,3), (0,3,2), and (0,3,3), the corresponding Q-values are also less than 3, but the target state still cannot be reached.

TABLE 3
Training Datasets of the Proposed DRAW

W^L	VM_{cur}	QoS_{cur}	VM_{obj}
$w(t_1)$	$vm^1_{t_1}, vm^2_{t_1}, \ldots, vm^n_{t_1}$	QoS_{t_1}	$VM^{obj}_{t_1}$
$w(t_2)$	$vm^1_{t_2}, vm^2_{t_2}, \ldots, vm^n_{t_2}$	QoS_{t_2}	$VM^{obj}_{t_2}$
...	...	...	...
$w(t_d)$	$vm^1_{t_d}, vm^2_{t_d}, \ldots, vm^n_{t_d}$	QoS_{t_d}	$VM^{obj}_{t_d}$

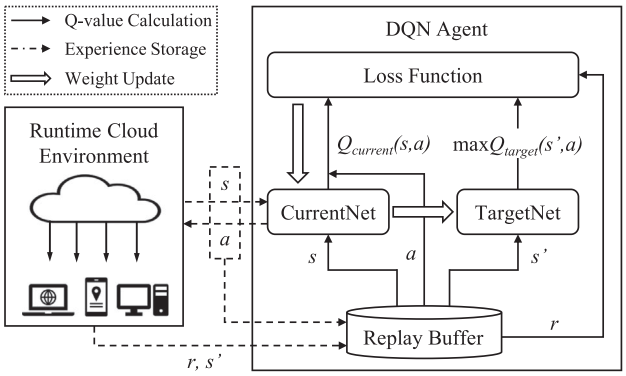

Fig. 3. Framework of the DQN.

TABLE 4
Q-Values of Different Management Operations When $W^L = (3396, 0.44, \ldots, 4500, 0.52)$

(a) $a = add_2$

No. of VMs		The 3 rd type				
		2	3	4	5	6
The 2nd type	2	2.1817	2.4086	2.4970	3.4825	2.6078
	3	2.2205	2.4800	2.5085	4.0620	2.9592
	4	3.2626	3.4202	**2.9373**	**2.7287**	3.0840
	5	3.2507	3.4231	**2.6141**	**2.7167**	3.0208
	6	3.1386	3.4099	3.4394	3.4404	2.5602
	7	2.1263	2.2184	2.3732	2.2790	1.5482
	8	-0.2792	-0.9395	-0.7040	-0.6493	-0.7305

(b) $a = add_3$

No. of VMs		The 3 rd type				
		2	3	4	5	6
The 2nd type	2	2.3901	2.4356	3.4386	3.4711	2.7635
	3	2.4623	2.4796	3.4914	3.5038	2.8162
	4	3.5993	3.6017	**2.8414**	**2.6509**	2.8389
	5	3.5966	3.6194	**2.7638**	**2.9144**	2.8215
	6	3.4516	3.5694	3.5889	3.5818	2.5742
	7	3.3957	3.4994	3.5182	3.5144	2.5268
	8	2.8262	2.9430	3.3425	3.3893	2.4268

(c) $a = rm_2$

No. of VMs		The 3 rd type				
		2	3	4	5	6
The 2nd type	2	2.2558	2.4186	3.2752	3.2130	2.5505
	3	2.4987	2.5315	3.3082	3.9283	3.4206
	4	3.2793	3.2864	**2.7842**	**2.8036**	3.5919
	5	3.3464	3.4839	**2.8070**	**2.6830**	3.5647
	6	3.0525	3.3991	4.2081	4.2761	4.0159
	7	3.0262	3.1143	3.7242	4.0933	2.8833
	8	2.2210	2.3671	2.5749	2.6139	2.4703

(d) $a = rm_3$

No. of VMs		The 3 rd type				
		2	3	4	5	6
The 2nd type	2	2.2554	2.5569	2.9865	2.8526	2.7999
	3	2.6383	2.9030	3.3954	3.6006	3.5939
	4	2.9357	3.0654	**2.8095**	**2.8324**	3.6123
	5	2.8534	3.1000	**2.9056**	**2.9801**	3.6996
	6	2.6886	2.7739	3.9891	4.1635	3.2388
	7	2.4021	2.6686	3.3323	3.7979	3.1625
	8	2.3506	2.4242	3.2886	3.3745	3.0897

- When the number of VMs is not within the assignable amount after taking a management operation (i.e., $getNext(VM_{cur}, a) \notin VM_{opt}$), the corresponding Q-value mutates into negative (the theoretical value should be -1), which will affect the Q-value prediction of the opposite management operations (e.g., add_2 and rm_2) for the same type of VMs. For example, when the number of VMs of the 2nd type is 8, the predicted Q-values caused by executing add_2 are all negative (theoretical value should be -1) because the current number of VMs of the 2nd type exceeds the maximum threshold. This will also affect the predicted Q-value of rm_2. Compared to the difference between the predicted Q-values of rm_3 when the number of VMs of the 2nd type is 7 and 8, this difference for rm_2 is much bigger. This is because the predicted Q-values of executing rm_2 are pulled low when the number of VMs of the 2nd type is 8, which will also affect the runtime decision-making of management operations. As shown in Table 4, when VMs are configured as (0,8,4), rm_2 should be chosen theoretically. However, the predicted Q-value of rm_2 (i.e., 2.5749) is relatively low due to the influence of add_2. At this time, add_3 with a higher Q-value (i.e., 3.3425) will be chosen, but the target state still cannot be reached.

To solve these issues, during the DQN training, the target Q-values (denoted by Q_target) are processed as

$$Q\_target = \begin{cases} I, & getNext(VM_{cur}, a) \notin VM_{opt} \\ 0, & VM_{cur} == VM_{obj} \\ \frac{1}{Q\_target}, & getNext(VM_{cur}, a) \in VM_{opt} \end{cases}. \quad (12)$$

If $getNext(VM_{cur}, a) \notin VM_{opt}$, this management operation will be regarded as illegal, and the corresponding Q_target will be indexed by I, which will not be stored in the replay buffer of the DQN. When $getNext(VM_{cur}, a) \in VM_{opt}$, $Q\_target = \frac{1}{Q\_target}$. When $VM_{cur} == VM_{obj}$, VM_{obj} is found and $Q\_target = 0$. During the training process, the DQN agent compares the Q-values of management operations and selects appropriate operations with smaller Q-values. Therefore, when approaching VM_{obj}, the value of Q_target will be smaller. By using this setting, the Q-value function will be continuous without singularity, and thus the target state can be reached through runtime decision-making. Table 5 shows the processed results of the Q-values in Table 4, where the target state is underlined for each of the four divided areas. Because there is a gap between the Q-values of management operations under the target state and other states, the termination condition is regarded to be met when Q-values are less than 0.15. For example, when VMs are configured as $(0, 2, 2)$, the target state that can meet the termination condition is $(0, 4, 4)$. Moreover, when the number of VMs of the 2nd type is 8, the predicted Q-values of rm_2 are no longer affected by add_2. Specially, when VMs are configured as $(0, 8, 4)$, the predicted Q-values of rm_2 and rm_3 are 0.2987 and 0.3756, respectively. At this time, rm_2 (i.e., the target management operation) can be chosen correctly.

The main steps of the proposed DQN-based prediction algorithm of management operations are as shown in Algorithm 1. First, the parameters of the DQN are randomly initialized (Line 3). For each data in the datasets (Line 4), the training will be performed. In each episode, the current state s_{cur} and the target state s_{obj} are initialized according to runtime datasets (Line 5). While $s_{cur} \neq s_{obj}$ (Line 6), an action a is randomly chosen with the probability of ε otherwise a is chosen with $\text{argmin}Q(s, a)$ (Line 8). Then, a is executed and the reward r is calculated according to Eq. (11)

TABLE 5
Processed Results of the Q-Values in Table 4

(a) $a = add_2$

No. of VMs		The 3 rd type				
		2	3	4	5	6
The 2nd type	2	0.2930	0.2849	0.2619	0.2824	0.3170
	3	0.2916	0.2825	0.2515	0.2650	0.3016
	4	0.2830	0.2845	**0.1405**	**0.1208**	0.2967
	5	0.2815	0.2844	**0.1251**	**0.1304**	0.2992
	6	0.2946	0.2848	0.2838	0.2838	0.3193
	7	0.2951	0.2916	0.2961	0.2894	0.3199
	8	I	I	I	I	I

(b) $a = add_3$

No. of VMs		The 3 rd type				
		2	3	4	5	6
The 2nd type	2	0.2755	0.2640	0.2839	0.2828	0.3099
	3	0.2831	0.2625	0.2610	0.2817	0.3076
	4	0.2786	0.2785	**0.1406**	**0.1473**	0.3067
	5	0.2787	0.2695	**0.1394**	**0.1337**	0.3074
	6	0.2834	0.2796	0.2789	0.2792	0.3186
	7	0.2853	0.2818	0.2812	0.2813	0.3209
	8	0.3106	0.3019	0.3070	0.3001	0.3400

(c) $a = rm_2$

No. of VMs		The 3 rd type				
		2	3	4	5	6
The 2nd type	2	0.3148	0.3065	0.2957	0.2918	0.3198
	3	0.3001	0.2979	0.2884	0.2687	0.3845
	4	0.2931	0.2898	**0.1452**	**0.1308**	0.2788
	5	0.2870	0.2824	**0.1382**	**0.1406**	0.2797
	6	0.2979	0.2852	0.2611	0.2593	0.2662
	7	0.3072	0.3035	0.2710	0.2697	0.3135
	8	0.3217	0.3108	0.2987	0.3070	0.3380

(d) $a = rm_3$

No. of VMs		The 3 rd type				
		2	3	4	5	6
The 2nd type	2	0.3350	0.3194	0.3005	0.3061	0.3083
	3	0.3156	0.3040	0.2953	0.2786	0.2788
	4	0.3026	0.2974	**0.1340**	**0.1403**	0.2782
	5	0.3060	0.2961	**0.1486**	**0.1420**	0.2755
	6	0.3387	0.3340	0.2971	0.2622	0.2909
	7	0.3563	0.3399	0.3075	0.2725	0.2937
	8	0.3950	0.3804	0.3756	0.3719	0.3905

(Line 9), and the next s' is generated (Line 10). If $s' \notin S_{opt}$ after executing a, this episode of training will be skipped (Lines 11~12). Otherwise, (s, a, r, s') will be stored into the replay buffer D (Line 14). If D is full, the oldest samples will be evicted [34]. Next, m samples are randomly drawn from D by the mini-batch and Q_target_j is calculated (Line 15). Then, the network weights ω are update by the loss function (Line 16), and ω' will be updated by ω every K iterations. Next, ω' is updated (Line 17), and the state transition between s_{cur} and s' happens (Line 18). Finally, the algorithm will be kept training until it converges.

Algorithm 1. DQN-Based Prediction of Management Operations

```
1:  Input: Runtime datasets described in Table 3
2:  Output: Prediction model of management operations
3:  Initialize Replay buffer $D$ with the capacity $M$, action-value function $Q$ with random weights $\omega$, and target action-value function $\hat{Q}$ with weights $\omega' = \omega$
4:  for each data in datasets do
5:    for each episode do
6:        Initialize the current state $s_{cur}$ and the target state $s_{obj}$ according to runtime datasets;
7:      while $s_{cur} \neq s_{obj}$ do
8:          Choose a random action $a$ with the probability of $\varepsilon$ otherwise $a$ is chosen with $\mathrm{argmin}Q(s, a)$;
9:          Execute $a$ and receive reward $r = R(s_{cur}, s_{obj}, a)$;
10:         Receive the next state $s' = getNext(s_{cur}, a)$;
11:         if $s' \notin S_{opt}$ then
12:           continue;
13:         else
14:           Store $(s, a, r, s')$ into $D$;
15:           Draw $m$ samples from $D$ by the mini-batch and calculate
              $Q\_target_j = \begin{cases} 0, & s_{cur} == s_{obj} \\ \frac{1}{r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \omega')}, & \text{other cases} \end{cases};$
16:           Update $\omega$ by the loss function $Loss = \frac{1}{m}\sum_{j=1}^{m} (Q\_target_j - Q(s, a; \omega))^2$;
17:         Update $\omega' = \omega$ every $K$ iterations;
18:         Update $s_{cur} = s'$;
19:         end
20:       end
21:     end
22: end
```

4.2 Runtime Decisions for Management Operations

At runtime, the decision-making of choosing management operations is executed, where the main steps are as shown in Algorithm 2. First, the DQN model is called to calculate the Q-values of all actions (i.e., management operations) and store them into the set Q_values (Line 9). According to Eq. (12), if an action $a \in A$ is illegal, its corresponding Q-value will be indexed by I (Lines 10~14). Therefore, VM_{opt} is used to judge whether VM_{cur} will be changed to an optional resource allocation plan after executing an action. Next, if the Q-values of each legal action $Q\_values[a] <= P$ (i.e., predefined threshold) or the resource allocation plan after executing an action has ever appeared (i.e., $VM\_List.contains(getNextVM(VM_{cur}, a))$, the objective resource allocation plan is found and the while loop can be broken (Lines 17~18). Otherwise, the action with the minimum Q-value will be executed (Line 20), and the objective resource allocation plan still needs to be explored. Meanwhile, VM_{cur} is stored in VM_List (Line 21), and the while loop will be continued.

Therefore, the objective resource allocation plan can be gradually found by this feedback-control mechanism.

TABLE 6
Configurations of Different Types of VMs

Config	Different types of VMs		
	Small	Medium	Large
CPU	1 core	1 core	1 core
Memory	1 GB	2 GB	4 GB
$Cost_L$	0.272 USD	0.291 USD	0.322 USD
$Cost_D$	0.068 USD	0.073 USD	0.080 USD

5 Performance Evaluation

This section first describes the experimental settings and datasets. Next, the proposed DRAW is evaluated with the consideration of the following research questions (RQs).

- *RQ 1:* Is the DRAW able to achieve adaptive resource allocation under various runtime cloud environments? (Section 5.2)
- *RQ 2:* How does the DQN-based prediction model of management operations perform? (Section 5.3)
- *RQ 3:* How much is the performance improvement of the DRAW compared to classic methods? (Section 5.4)

For *RQ 1*, the performance of the DRAW is comparable with the ideal one under various scenarios, where the performance gap is less than 5%. For *RQ 2*, the DQN-based prediction model of management operations can achieve the highest prediction accuracy of management operations (i.e., 90.69%) when the Q-value threshold is set to 0.15. For *RQ 3*, the DRAW outperforms three classic methods by 3~13% under different scenarios.

5.1 Experimental Settings and Datasets

We conduct experiments on CloudStack [36], which contains three types of VMs (i.e., small, medium, and large) referring to Alibaba Cloud,[1] where the corresponding numbers of VMs are vm_S, vm_M, and vm_L, respectively. Table 6 shows their configurations that consist of CPUs, memories, $Cost_L$, and $Cost_D$. The current resource allocation plan is represented by $VM_{current} = (vm_S, vm_M, vm_L)$.

Next, the RUBiS [37] is performed, which is a prototype of auction sites that models *eBay.com*. The RUBiS provides clients to emulate user behaviors (e.g., browsing, bidding, and rating) for various workload patterns, where the number of clients indicates the amount of workloads. Assuming that a workload has two types of user behaviours, where Behaviour A represents browsing and Behaviour B represents bidding and rating. The Google cluster-usage traces [38] are used as the workload datasets, whose amount is mapped to $[3000, 5000]$ and the proportions of Behaviours A and B are randomly distributed in $[0, 1]$ (the sum of these two behaviours' proportions equals 1).

During the process of resource allocation, we assume that the time length of predictable workloads is 90 min (this value is not fixed but can be adjusted according to demands) and VM configurations are adjusted every 15 min (because it takes time for VM switching, system stabilization, and QoS monitoring after adjusting VM configurations, and thus the next adjustment can only be executed after a period of time). To balance the QoS and resource costs, r_1 and r_2 are set to 320 and 10 [39], respectively. Moreover, the DQN algorithm is implemented by using TensorFlow 1.4.0 [40], where the number of DNN layers is 4 (the ratio of the number of neurons in each layer is 16: 512: 512: 6). The learning rate, the discount factor γ, the probability of exploration ε, the number of episodes, and the capacity of replay buffer are set to 0.001, 0.8, 0.1, 500, and 2000, respectively. We conduct model training on a workstation configured with Intel Xeon E5-2630 v3 CPU*2 and 32 G memory, and the prediction model of management operation is obtained after about 15 hours of training.

Algorithm 2. Runtime Decision-Making for Management Operations

```
1:  Input: The current workload in a workload-time window
        $W^L$, and the current resource allocation plan $VM_{cur}$ with
        the corresponding $QoS_{cur}$
2:  Declare:
3:    $A$ – the action space, where $a \in A$
4:    $a\_List$ – the set of $a$ with the minimum Q-value
5:    $Q\_values$ – the set of Q-values of all actions
6:    $getQvalueByDQN()$ – call the DQN model to calculate Q-
      values
7:    $VM\_List$ – the set of resource allocation plans that occur
      during the decision-making process
8:  while true do
9:    $Q\_values = getQvalueByDQN(W^L, VM_{cur}, QoS_{cur})$;
10:   for each $a \in A$ do
11:     if $getNext(VM_{cur}, a) \notin VM_{opt}$ then
12:       $Q\_values[a] = I$;
13:     end
14:   end
15:   $a\_List = \text{argmin}_a(Q\_values)$;
16:   $a = a\_List.get\_Action\_Random()$;
17:       if (for each $Q\_values[a] <= P$ || $Q\_values[a] ==$
          $I$ || $VM\_List.contains(getNextVM(VM_{cur}, a))$) then
18:     break;
19:   else
20:     $VM_{cur} = Execute(VM_{cur}, a)$;
21:     $VM\_List.add(VM_{cur})$;
22:   end
23: end
```

On the CloudStack-based private cloud, Google cluster-usage traces are used as the workload data (700 hours). The system runtime data is collected as the experimental datasets,[2] which are split into the training set (50%, used for training the model), the validation set (25%, used for tuning hyper-parameters), and the testing set (25%, used for testing the model). Next, we select the data of five scenarios (6 hours per scenario) from Google cluster-usage traces as typical workloads with the consideration of different workload patterns such as growing, cyclic bursty, and random, as shown in Fig. 4. Based on the CloudStack-based private cloud, the DRAW is used to make runtime decisions of resource allocation in each scenario, where the threshold P is set to 0.15 according to expert experience. When the Q-

1. https://help.aliyun.com/document_{d}etail/25378.html
2. https://github.com/yangljian/DRAW

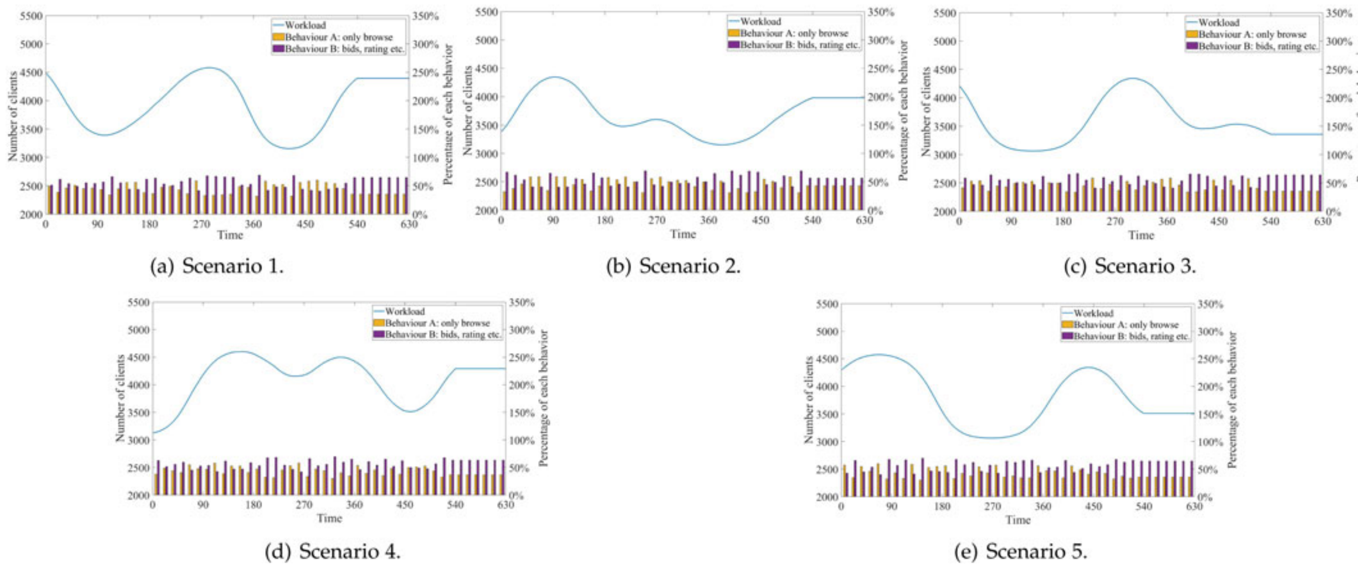

Fig. 4. Different scenarios of resource allocation with workload-time windows.

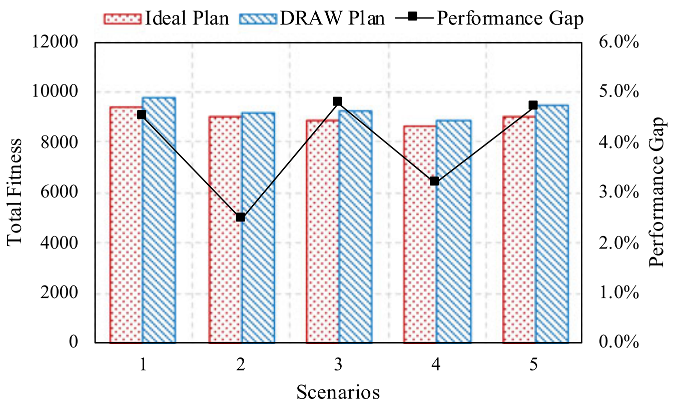

Fig. 5. Comparison of the total fitness between the DRAW and ideal results under various scenarios.

values of all management operations are less than or equal to P, no more management operation will be taken.

5.2 RQ 1: Effectiveness of the DRAW

First, the effectiveness of the proposed DRAW is evaluated under various scenarios illustrated in Fig. 4. Specifically, the performance of the DRAW is compared with the ideal results under these scenarios. In practice, it is hard to obtain the ideal resource allocation plan because all possibilities are required to be investigated, which results in extremely high complexity. Under different cases, we combine management experience and small-scale verification to obtain the corresponding ideal plans. As shown in Fig. 5, the DRAW obtains equivalent performance with the ideal results in terms of the total fitness under various scenarios, where the performance gap is less than 5%. As shown in Fig. 6, taking Scenario 2 as an example, the DRAW can achieve similar resource allocation plans as the ideal ones only with small differences. Especially, they are the same most of the time (i.e., 15, 30, 75, 150, 165, and 240). The results present that the DRAW is able to obtain optimal/near-optimal performance. This is because the DRAW can learn management knowledge from the historical data of operating systems and then utilize the knowledge to perform effective resource management in real-world environments.

Specifically, we use Scenario 2 at 60 min as an instance to describe the resource allocation process of the DRAW, where $W^L = (3404, 0.45, \ldots, 3774, 0.44)$. As shown in Table 7, when VM_{cur} is initialized as $(1, 0, 5)$, add_2 is with the smallest predicted Q-value among all operations. Therefore, the number of VMs of the 2nd type is added by one, and VM_{cur} becomes

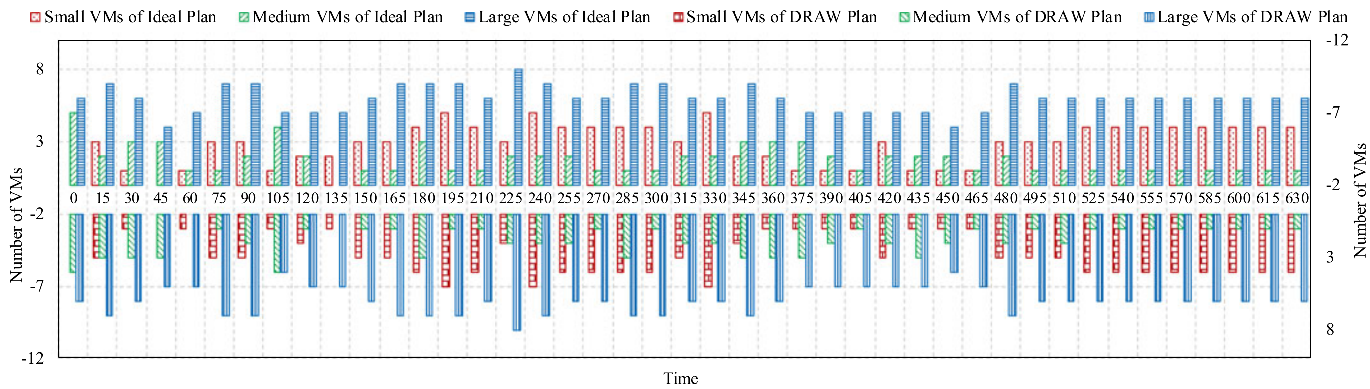

Fig. 6. Comparison of resource allocation plans between the DRAW and ideal ones under Scenario 2.

TABLE 7
Resource Allocation Process of the DRAW in Scenario 2 at 60 Min

VM_{cur}		add_1	rm_1	add_2	rm_2	add_3	rm_3
0	$1,0,5$	0.486	0.480	**0.426**	0.431	0.486	0.479
1	$1,1,5$	**0.321**	0.362	0.377	0.334	0.406	0.339
2	$2,1,5$	**0.231**	0.269	0.276	0.264	0.286	0.292
3	$3,1,5$	0.218	0.249	0.254	0.249	**0.209**	0.285
4	$3,1,6$	0.183	0.173	0.168	0.155	**0.150**	0.170
5	$3,1,7$	0.147	0.140	0.137	0.140	0.120	0.149

$(1, 1, 5)$. Next, add_1 leads to the smallest predicted Q-value, and a VM of the 1st type is added. For each step, the operation with the smallest predicted Q-value will be chosen. When $VM_{cur} = (3, 1, 7)$, the predicted Q-values of all operations have satisfied the threshold P (i.e., less than 0.15), which indicates that the objective resource allocation plan has been found and there is no need to take any operations.

5.3 RQ 2: Performance of the DQN-Based Prediction Model for Choosing Management Operations

First, we study the correctness of choosing management operations (reflects the accuracy of the Q-value prediction model) with different numbers of steps to the objective plan, measured by the action accuracy rate (AAR), which is defined as

$$AAR = \frac{A}{M} \times 100\%, \tag{13}$$

where M is the number of all management operations that have been taken, and A is the number of correct ones. If the current resource allocation plan can be close to the objective one after executing management operations, these management operations are considered to be correct and thus the corresponding predicted Q-values are accurate.

As shown in Fig. 7, the AAR slightly declines when approaching the objective plan. Specifically, the AAR is around 93% when there are seven or more steps from the current plan to the objective one. Even if the current plan almost matches the objective one, the AAR can remain

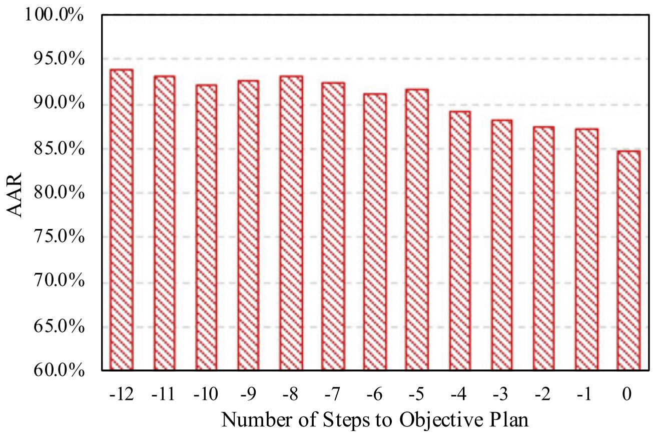

Fig. 7. Correctness of choosing management operations with different numbers of steps to the objective plan.

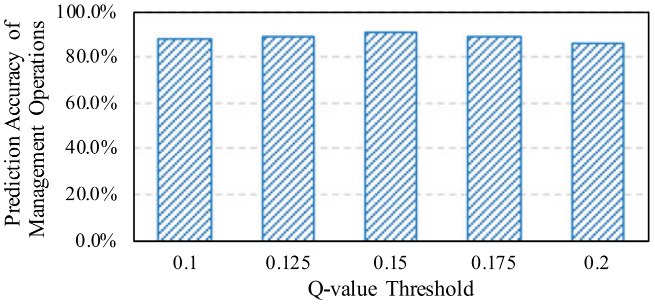

Fig. 8. Prediction accuracy of management operations with different Q-value thresholds.

about 85%. Thus, when there are many steps from the current plan to the objective one, the high correctness of choosing management operations can be well promised with the support of the DQN-based prediction model for management operations. Only when approaching the objective plan, some small variations happen in the correctness of choosing management operations.

Next, we study the impact of the threshold P on finding the objective plan VM_{obj}. Different settings of P may affect the accuracy of the DQN-based prediction model for management operations. As shown in Fig. 8, P is set to 0.1, 0.125, 0.15, 0.175, 0.2, respectively. Accordingly, the accuracy can reach 87.81%, 88.79%, 90.69%, 88.75%, and 86.41%, respectively. Therefore, when P is set to 0.15, the model can achieve the highest prediction accuracy. When P is too low, even if VM_{obj} is found, there is still a situation where some predicted Q-values are greater than P, which will cause the continued resource adjustment. For example, as shown in Table 8, when P is set to 0.125, the predicted Q-value of add_2 (i.e., 0.127) is greater than P even if $VM_{obj} = (2, 3, 7)$ is found, and thus add_2 will still be executed. On the contrary, when P is too high, although the objective plan has not been found, the resource adjustment still ends early because all the predicted Q-values are less than P. For example, as shown in Table 9, P is set to 0.175. When $VM_{cur} = (2, 2, 7)$, the predicted Q-values of all operations are already less than P. Therefore, no more operation will be executed and $VM_{obj} = (3, 4, 7)$ cannot be found. According to the evaluations by using training datasets, 0.15 is used as the Q-value threshold in our experiments.

5.4 RQ 3: Performance Improvement of the DRAW

The performance of the proposed DRAW is compared with the classic ML-based [23], DQN-based [34], and rule-based methods. When evaluating these methods, whether to

TABLE 8
An Example When $P = 0.125$ and $VM_{obj} = (2, 3, 7)$

VM_{cur}		add_1	rm_1	add_2	rm_2	add_3	rm_3
0	$5,2,6$	0.343	0.325	0.328	0.313	**0.298**	0.378
1	$5,2,7$	0.292	0.324	**0.251**	0.329	0.328	0.303
2	$5,3,7$	0.311	**0.245**	0.308	0.252	0.323	0.262
3	$4,3,7$	0.237	**0.193**	0.203	0.231	0.276	0.248
4	$3,3,7$	0.219	**0.166**	0.197	0.189	0.174	0.194
5	$2,3,7$	0.138	0.128	**0.127**	0.142	0.135	0.147

TABLE 9
An Example When $P = 0.175$ and $VM_{obj} = (3, 4, 7)$

	VM_{cur}	add_1	rm_1	add_2	rm_2	add_3	rm_3
0	$1, 1, 5$	0.358	0.386	0.324	0.342	**0.317**	0.332
1	$1, 1, 6$	0.305	0.332	0.336	0.315	**0.288**	0.303
2	$1, 1, 7$	0.280	0.266	**0.257**	0.294	0.272	0.266
3	$1, 2, 7$	**0.179**	0.211	0.222	0.210	0.185	0.207
4	$2, 2, 7$	0.170	0.172	0.171	0.161	0.169	0.155
5	$3, 2, 7$	0.159	0.167	0.163	0.156	0.147	0.134
6	$3, 3, 7$	0.122	0.121	0.122	0.132	0.126	0.133
7	$3, 4, 7$	0.114	0.123	0.108	0.124	0.123	0.116

TABLE 10
Rule Settings Without Workload-Time Windows

Rule	Operation
RT $>$ 1.4 s	vm_L is added by one
1.2 s $<$ RT $\leq$ 1.4 s	vm_M is added by one
1.0 s $<$ RT $\leq$ 1.2 s	VMs remain unchanged
0.8 s $<$ RT $\leq$ 1.0 s	vm_M is removed by one
RT $\leq$ 0.8 s	vm_L is removed by one

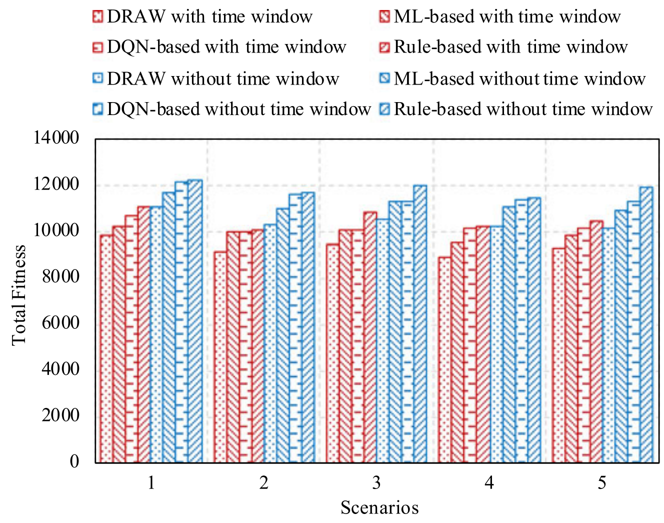

Fig. 9. Comparison among different methods in total fitness.

introduce the workload-time windows is considered respectively. Specifically, the ML-based method without workload-time windows uses particle swarm optimization (PSO) to explore the objective plan based on the current workload, while the one with workload-time windows can explore the objective plan in multiple workload-time windows. Similarly, the DQN-based method without workload-time windows only considers the objective plan under the current workload, while the one with workload-time windows considers both the current and future workloads. The rule-based method without workload-time windows uses the current RT to set rules for choosing operations, while the one with workload-time windows combines RT and workload change rate (i.e., $\frac{\overline{W^L}}{W_{cur}}$) to set compound rules for choosing operations. The detailed rule settings are given in Tables 10 and 11.

As shown in Fig. 9, the DRAW outperforms other classic methods in terms of the total fitness under five different scenarios. For the scenarios with workload-time windows, the DRAW outperforms the rule-based method by 9∼13%. This is because the rule-based method involves segmentation rules set by experts, but it is infeasible to make a rule well applied to the resource allocation problem in the complex cloud environments with dynamic workloads. Moreover, the DRAW achieves better performance than the ML-based method by 3∼8%, where the QoS prediction model in the ML-based method is trained by using the same datasets as the DRAW. The DRAW only needs to compare the Q-values of management operations when selecting appropriate operations instead of accurately predicting all the Q-values. Therefore, the proposed DQN agent can achieve high prediction accuracy of choosing management operations without overly relying on massive training data. In contrast, the ML-based method requires massive historical data for building an accurate QoS prediction model. Without adequate data support, this method cannot achieve efficient resource allocation due to inaccurate QoS prediction. When compared with the DQN-based method, the DRAW reduces the total fitness by 6∼12% under different scenarios. In the DQN-based method, during the training of the Q-value prediction model of management operations, there exist mutation issues of Q-values (detailed descriptions are given in Section 4.1). Therefore, it is hard to inaccurately predict the Q-values under some boundary conditions, which would

TABLE 11
Rule Settings With Workload-Time Windows

Rule 1	Rule 2	Operation
$\frac{\overline{W^L}}{W_{cur}} > 1.1$	RT $>$ 1.2 s	vm_L is added by one
	1.0 s $<$ RT $\leq$1.2 s	vm_M is added by one
	RT $\leq$1.0 s	VMs remain unchanged
$0.9 < \frac{\overline{W^L}}{W_{cur}} \leq 1.1$	RT $>$ 1.4 s	vm_L is added by one
	1.2 s $<$ RT $\leq$ 1.4 s	vm_M is added by one
	1.0 s $<$ RT $\leq$ 1.2 s	VMs remain unchanged
	0.8 s $<$ RT $\leq$ 1.0 s	vm_M is removed by one
	RT $\leq$ 0.8 s	vm_L is removed by one
$0 < \frac{\overline{W^L}}{W_{cur}} \leq 0.9$	RT $>$ 1.2 s	VMs remain unchanged
	1.0 s $<$ RT $\leq$ 1.2 s	vm_M is removed by one
	RT $\leq$ 1.0 s	vm_L is removed by one

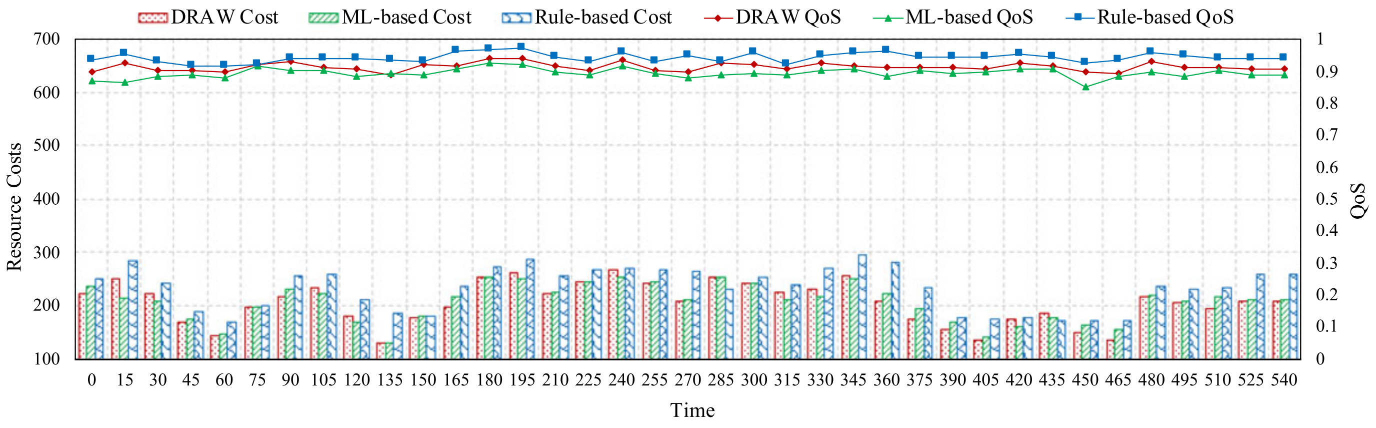

Fig. 10. Comparison among different methods in QoS and resource costs.

TABLE 12
Runtime Execution Time (s) of Different Methods Under Various Scenarios

Scenarios	Different methods		
	DRAW	ML-based	Rule-based
1	0.35	218	0.14
2	0.48	201	0.11
3	0.33	217	0.18
4	0.56	230	0.13
5	0.44	194	0.12

lead to inefficient resource allocation. Furthermore, compared to the scenarios without workload-time windows, through introducing workload-time windows, the performance of the DRAW, ML-based, DQN-based, and rule-based methods can be improved by 10.77%, 11.26%, 11.60%, and 11.00%, respectively. This is because grasping the trend of future workload changes plays an important role in improving the efficiency of resource allocation.

More specifically, we present the performance of the DRAW in a workload-time window from the perspectives of QoS and resource costs. As shown in Fig. 10, the rule-based method always leads to the best QoS, while the ML-based method results in the worst QoS most of the time. By contrast, the QoS produced by the DRAW is between the others. Although the rule-based method is with the best QoS, it causes the highest resource costs. This is because it only sets rules for optimizing RT but neglects the issue of resource costs. However, it is difficult to set rules for meeting various service requirements. In contrast, the DRAW can make a better trade-off between QoS and resource costs. Most of the time, the DRAW leads to fewer resource costs compared to the other methods, while the corresponding QoS is also good. Meanwhile, the DRAW is able to keep steady QoS (around 0.91) at different times. By contrast, the ML-based method cannot keep steady QoS, whose values constantly change between 0.85 and 0.91.

Finally, we compare the runtime execution time of different methods including the DRAW, ML-based, and rule-based methods under various scenarios with the consideration of workload-time windows. As shown in Table 12, both the DRAW and rule-based methods are able to find resource allocation plans in milliseconds under different scenarios, but the rule-based method cannot well guarantee the effectiveness of resource allocation (as analyzed in Figs. 9 and 10). By contrast, it needs to take several minutes to search for resource allocation plans when using the ML-based method, which cannot well meet the real-time requirements of system management.

6 Conclusion

In this paper, we propose a DRAW to explore the adaptive resource allocation for cloud-based software services. First, we design a DQN-based prediction model of management operations by introducing workload-time windows, which is able to predict appropriate management operations under different system states. Next, we develop a feedback-control mechanism to find the objective resource allocation plan under the current system state by iteratively executing management operations. Using the RUBiS benchmark, we conduct extensive simulation experiments to validate the effectiveness of the proposed DRAW in realizing optimal/near-optimal performance of cloud resource allocation. Specifically, the performance gap between the DRAW and the ideal one is less than 5%. Moreover, the prediction accuracy of management operations can reach 90.69% by the DRAW. Besides, the DRAW outperforms other classic methods by 3∼13% under different scenarios with various workloads and service requests.

In our future work, we will further evaluate and improve the performance of the proposed method in real-world cloud production environments with unseen scenarios. Moreover, we plan to further explore the feasibility of other DRL variations (e.g., actor-critic, policy gradient, etc.) on addressing the problem of cloud resource allocation.

References

[1] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[2] M. Shojafar, N. Cordeschi, and E. Baccarelli, "Energy-efficient adaptive resource management for real-time vehicular cloud services," *IEEE Trans. Cloud Comput.*, vol. 7, no. 1, pp. 196–209, Jan.–Mar 2019.
[3] A. Jindal, K. Karanasos, S. Rao, and H. Patel, "Selecting subexpressions to materialize at datacenter scale," in *Proc. 44th Int. Conf. Very Large Data Bases*, 2018, pp. 800–812.
[4] W. A. Simm *et al.*, "SE in ES: Opportunities for software engineering and cloud computing in environmental science," in *Proc. 40th Int. Conf. Softw. Eng.: Softw. Eng. Soc.*, 2018, pp. 61–70.
[5] T. Chen and R. Bahsoon, "Self-adaptive and online QoS modeling for cloud-based software services," *IEEE Trans. Softw. Eng.*, vol. 43, no. 5, pp. 453–475, May 2017.

[6] M. J. Farooq and Q. Zhu, "QoE based revenue maximizing dynamic resource allocation and pricing for fog-enabled mission-critical IoT applications," *IEEE Trans. Mobile Comput.*, vol. 20, no. 12, pp. 3395–3408, Dec. 2021.

[7] D. Ardagna, M. Ciavotta, R. Lancellotti, and M. Guerriero, "A hierarchical receding horizon algorithm for QoS-driven control of multi-IaaS applications," *IEEE Trans. Cloud Comput.*, vol. 9, no. 2, pp. 418–434, Apr.–Jun 2021.

[8] P. Haratian, F. Safi-Esfahani, L. Salimian, and A. Nabiollahi, "An adaptive and fuzzy resource management approach in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 907–920, Oct.–Dec 2019.

[9] L. Wei, C. H. Foh, B. He, and J. Cai, "Towards efficient resource allocation for heterogeneous workloads in IaaS clouds," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 264–275, Jan.–Mar 2018.

[10] N. Liu *et al.*, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 372–382.

[11] Z. Chen, J. Hu, G. Min, C. Luo, and T. El-Ghazawi, "Adaptive and efficient resource allocation in cloud datacenters using actor-critic deep reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1911–1923, Aug. 2022.

[12] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Oct.–Dec 2015.

[13] F.-H. Tseng, X. Wang, L.-D. Chou, H.-C. Chao, and V. C. M. Leung, "Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1688–1699, Jun. 2018.

[14] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. El-Ghazawi, "Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 923–934, Apr. 2020.

[15] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey, "Forecasting cloud application workloads with CloudInsight for predictive resource management," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2020.2998017.

[16] M. Abdullah, W. Iqbal, A. Mahmood, F. Bukhari, and A. Erradi, "Predictive autoscaling of microservices hosted in fog microdata center," *IEEE Syst. J.*, vol. 15, no. 1, pp. 1275–1286, Mar. 2021.

[17] D. Buchaca, J. L. Berral, C. Wang, and A. Youssef, "Proactive container auto-scaling for cloud native machine learning services," in *Proc. 13th Int. Conf. Cloud Comput.*, 2020, pp. 475–479.

[18] S. Khatua, P. K. Sur, R. K. Das, and N. Mukherjee, "Heuristic-based resource reservation strategies for public cloud," *IEEE Trans. Cloud Comput.*, vol. 4, no. 4, pp. 392–401, Oct.–Dec 2016.

[19] Z. Chen, L. Yang, Y. Huang, X. Chen, X. Zheng, and C. Rong, "PSO-GA-based resource allocation strategy for cloud-based software services with workload-time windows," *IEEE Access*, vol. 8, pp. 151500–151510, 2020.

[20] R. Tolosana-Calasanz, J. Diaz-Montes, O. F. Rana, and M. Parashar, "Feedback-control & queueing theory-based resource management for streaming applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1061–1075, Apr. 2017.

[21] M. R. HoseinyFarahabady, J. Taheri, A. Y. Zomaya, and Z. Tari, "A dynamic resource controller for resolving quality of service issues in modern streaming processing engines," in *Proc. 19th Int. Symp. Netw. Comput. Appl.*, 2020, pp. 1–8.

[22] E. Makridis, K. Deliparaschos, E. Kalyvianaki, A. Zolotas, and T. Charalambous, "Robust dynamic CPU resource provisioning in virtualized servers," *IEEE Trans. Services Comput.*, vol. 15, no. 2, pp. 956–969, Mar./Apr. 2022.

[23] X. Chen, J. Lin, B. Lin, T. Xiang, Y. Zhang, and G. Huang, "Self-learning and self-adaptive resource allocation for cloud-based software services," *Concurrency Comput., Pract. Experience*, vol. 31, no. 23, 2019, Art. no. e4463.

[24] J. Wang *et al.*, "A machine learning framework for resource allocation assisted by cloud computing," *IEEE Netw.*, vol. 32, no. 2, pp. 144–151, Mar./Apr. 2018.

[25] S. Choochotkaew, H. Yamaguchi, T. Higashino, D. Schäfer, J. Edinger, and C. Becker, "Self-adaptive resource allocation for continuous task offloading in pervasive computing," in *Proc. Int. Conf. Pervasive Comput. Commun. Workshops*, 2018, pp. 663–668.

[26] Z. Chen, J. Hu, and G. Min, "Learning-based resource allocation in cloud data center using advantage actor-critic," in *Proc. 53rd Int. Conf. Commun.*, 2019, pp. 1–6.

[27] A. Alsarhan, A. Itradat, A. Y. Al-Dubai, A. Y. Zomaya, and G. Min, "Adaptive resource allocation and provisioning in multi-service cloud environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 1, pp. 31–42, Jan. 2018.

[28] A. I. Orhean, F. Pop, and I. Raicu, "New scheduling approach using reinforcement learning for heterogeneous distributed systems," *J. Parallel Distrib. Comput.*, vol. 117, pp. 292–302, 2018.

[29] X. Chen, F. Zhu, Z. Chen, G. Min, X. Zheng, and C. Rong, "Resource allocation for cloud-based software services using prediction-enabled feedback control with reinforcement learning," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2020.2992537.

[30] W. Guo, W. Tian, Y. Ye, L. Xu, and K. Wu, "Cloud resource scheduling with deep reinforcement learning and imitation learning," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3576–3586, Mar. 2021.

[31] B. Wang, F. Liu, and W. Lin, "Energy-efficient VM scheduling based on deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 125, pp. 616–628, 2021.

[32] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[33] I. V. Paputungan, A. F. M. Hani, M. F. Hassan, and V. S. Asirvadam, "Real-time and proactive SLA renegotiation for a cloud-based system," *IEEE Syst. J.*, vol. 13, no. 1, pp. 400–411, Mar. 2019.

[34] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[36] N. Sabharwal, *Apache CloudStack Cloud Computing*. Birmingham, U.K.: Packt Publishing Ltd, 2013.

[37] RUBiS: Rice University Bidding System Benchmark, 2013. Accessed: Aug. 26, 2020. [Online]. Available: http://rubis.ow2.org/

[38] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format schema," *Google White Paper*, pp. 1–14, 2011.

[39] X. Chen, H. Wang, Y. Ma, X. Zheng, and L. Guo, "Self-adaptive resource allocation for cloud-based software services based on iterative QoS prediction model," *Future Gener. Comput. Syst.*, vol. 105, pp. 287–296, 2020.

[40] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th Symp. Oper. Syst. Des. Implementation*, 2016, pp. 265–283.

Xing Chen (Member, IEEE) received the BS and PhD degrees from Peking University, Beijing, China, in 2008 and 2013, respectively. Upon completion of the PhD degree, he joined Fuzhou University and has held the rank of professor since 2020. Now, he is the deputy director with the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, and leads the Systems Research Group. His research focuses on the software systems and engineering approaches for cloud and mobility. His current projects cover the topics from self-adaptive software, computation offloading, model driven approach and so on. He has published more than fifty journal and conference articles, and was awarded two First Class Prizes for Provincial Scientific and Technological Progress, separately in 2018 and 2020.

Lijian Yang received the BS degree in computer science from Fujian Normal University, Fujian, China, in 2019. He is currently working toward the MS degree in computer technology with the College of Computer and Data Science, Fuzhou University, Fuzhou, China. His current research interests include system software, edge computing, and cloud computing.

Zheyi Chen received the MSc degree in computer science and technology from Tsinghua University, Beijing, China, in 2017, and the PhD degree in computer science from the University of Exeter, Exeter, U.K., in 2021. He is currently an assistant professor and Qishan scholar with the College of Computer and Data Science, Fuzhou University, China. His research interests include cloud-edge computing, resource optimization, deep learning, and reinforcement learning. He has published more than 10 research papers in reputable international journals and conferences such as the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Industrial Informatics*, and IEEE ICC.

Geyong Min (Member, IEEE) received the BSc degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 1995, and the PhD degree in computing science from the University of Glasgow, Glasgow, U.K., in 2003. He is a professor of high performance computing and networking with the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences, University of Exeter, U.K. His research interests include future Internet, computer networks, wireless communications, multimedia systems, information security, high-performance computing, ubiquitous computing, modelling, and performance engineering.

Xianghan Zheng received the MSc degree in distributed system and the PhD degree in information communication technology from the University of Agder, Kristiansand, Norway, in 2007 and 2011, respectively. He is currently a professor with the College of Computer and Data Science, Fuzhou University, China. His current research interests include new generation network with a special focus on cloud computing services and applications, and Big Data processing and security.

Chunming Rong (Senior Member, IEEE) is a professor and the head of the Center for IP-based Service Innovation (CIPSI), University of Stavanger (UiS), Norway. He is the chair of IEEE Cloud Computing and an executive member of Technical Consortium on High Performance Computing (TCHPC) and the chair of STC on Blockchain in IEEE Computer Society, and served as global co-chair of IEEE Blockchain in 2018. He is also advisor of the StandICT.EU to support European scandalization activities in ICT. He is also co-founder of two start-ups bitYoga and Dataunitor in Norway, both received EU Seal of Excellence Award in 2018. He was adjunct senior scientist leading Big-Data Initiative at NORCE (2016–2019), the vice president of CSA Norway Chapter (2016–2017). His research work focuses on cloud computing, data analytics, cyber security, and blockchain. He is honoured as member of the Norwegian Academy of Technological Sciences (NTVA) since 2011. He has extensive contact network and projects in both the industry and academic. He is also founder and Steering chair of IEEE CloudCom conference and workshop series. He is co-editors-in-chief of the *Journal of Cloud Computing* (ISSN: 2192-113X) by Springer, has served as the steering chair (2016–2019), steering member and associate editor of the *IEEE Transactions on Cloud Computing* (TCC) since 2016. He has supervised 26 PhDs, nine Post-Docs and more than 60 master projects. He has extensive experience in managing large-scale R&D projects, both in Norway and EU.