RADEAN: A Resource Allocation Model Based on Deep Reinforcement Learning and Generative Adversarial Networks in Edge Computing

Zhaoyang Yu[1,2], Sinong Zhao[1], Tongtong Su[1], Wenwen Liu[1], Xiaoguang Liu[1(✉)], Gang Wang[1], Zehua Wang[2], and Victor C. M. Leung[2]

[1] College of Computer Science, ICIC, TMCC, SysNet, DISSec, GTIISC, Nankai University, Tianjin, China
{yuzz,zhaosn,sutt,liuww,liuxg,wgzwp}@nbjl.nankai.edu.cn
[2] Department of Electrical and Computer Engineering, WiNMos Lab, University of British Columbia, Vancouver, Canada
{zwang,vleung}@ece.ubc.ca

Abstract. Edge computing alleviates the network congestion and latency pressure on the remote cloud as well as the computation stress on end devices. However, facing numerous tasks, how to effectively allocate and manage the resources of edge servers is of great significance, which affects both the benefits of service providers and Quality of Service (QoS) of users. This paper proposes a resource allocation model based on Deep Reinforcement Learning (DRL) and Generative Adversarial Networks (GAN)(RADEAN). It considers the future resource occupancy of edge servers, and applies multi-replay memory with priority to eliminate the interaction between experiences and improve sampling efficiency. We maximize the average resource utilization of edge servers while ensuring the average transmission latency (ATL) and average execution time (AET) of tasks in a long-term view. Specifically, based on the state which consists the predicted resource occupation output by GAN, the current resource usage status of edge servers and the characteristics of the task queue, DRL agent makes resource allocation decision for each task. We conduct experiments using real-world data trace, and show that RADEAN outperforms traditional and state-of-the-art models with great generalization, reaching the maximum performance improvement of 48.21% compared with MMRA. The ATL and AET of tasks are also presented to reflect the QoS guarantee. Ablation experiments prove the effectiveness of multi-replay memory and priority sub-modules.

Keywords: edge computing · resource allocation · deep reinforcement learning · generative adversarial networks

This research is supported in part by the NSF of China (62272253, 62272252, 62141412) and Fundamental Research Funds for the Central University, China Scholarship Council (CSC). The first author is supported by CSC (Grant No. 202106200061) as a visiting Ph.D. student at the University of British Columbia, Canada under the supervision of Prof. Victor C.M. Leung.

A. Zaslavsky et al. (Eds.): MobiQuitous 2023, LNICST 593, pp. 257–277, 2024.
https://doi.org/10.1007/978-3-031-63989-0_13

1 Introduction

Recently, the number of Internet of Things (IoT) terminals increases exponentially and is expected to reach 125 billion by 2030 [1]. Substantial delay-sensitive and computing-intensive tasks, such as Augmented Reality (AR), Virtual Reality (VR) and target detection, bring huge challenges to task execution correspondingly. Multiple edge clouds cooperate for processing tasks, reducing task execution delay, data transmission security and backhaul impact in remote cloud and the computing pressure on end devices.

An overloaded CPU will cause long execution delay, which is unacceptable for latency-sensitive tasks. Compared with the remote cloud, the conditions of computing resource allocation in edge are more complicated. First, the computation resource capacities, storage space and processing speed of edge servers are relatively limited. Second, the heterogeneity and dynamics of end-user devices and edge nodes determine that the resource allocation in edge is NP-hard [2]. User tasks from various hardware specifications with different objectives, architectures, and communication protocols, etc. Edge clouds with various coverage areas are equipped with different edge servers, conducting diverse strategies of resource allocation. Third, the mobility of users between multiple edge clouds involves the release and reallocation of resources. Putting these all together creates a more serious challenge for resource allocation in edge.

For each task submitted by users, the resource allocation manager in the edge server decides how many computing resources are going to be assigned to this task and how long it is allowed to occupy resources. If the task is allocated few computing resources, its increased occupation time may cause task congestion. The waiting and completion delay of tasks will raise accordingly, which is potentially fatal for latency-sensitive tasks. However, assigning too many computing resources to a certain task may result in resource deficit for others. In this case, these tasks must be migrated to other edge clouds or even remote clouds for execution. Task migration brings extensive transmission time, protracted execution latency and high migration cost, thereby reducing the Quality of Service (QoS) for users. Therefore, it's an urgent problem that how to allocate and schedule resources in edge servers to achieve high resource utilization, improve the efficiency of service providers, and guarantee users' QoS in the meantime.

Some previous works involved in resource allocation focused on the cost [3,4], energy consumption [5] and latency [6]. The research with the optimization goal of resource utilization [7] ignored the impact of future resource usage of edge servers. Applying Deep Reinforcement Learning (DRL) in resource management [8,9] makes up for the shortcoming of the high computation cost of traditional approaches such as heuristic [10] and meta-heuristic [3,11] algorithms. Most DRL methods enhance the performance by improving the model structure during training process, but ignore the optimization from the perspective of task characteristic analysis. In fact, a job contains multiple tasks with dependencies, and the generation and requests of tasks are correlated with time series to a certain extent. Therefore, facing the dynamic and predictable nature of task offloading, the resource utilization of edge servers can be predicted based on the

historical system data, so as to make better resource allocation decisions. The semi-supervised method Generative Adversarial Networks (GAN) performs confrontational training without requiring a large number of training samples and prior knowledge. It generates clearer data and establishes description model to enhance the prediction effect, behaving well for time series prediction in workload forecast [12], financial prediction [13] and other fields. Hence, we choose GAN to estimate the future resource condition and assist the resource allocation decisions for DRL agent. Moreover, we build an improved DRL approach to adjust the resource allocation strategy through the trial-return feedback mechanism. It tracks the obtained reward for resource allocation decisions and feeds back to the agent for policy adjustment, which is more robust and adaptable in edge to achieve the long-term maximum average resource utilization.

Taking the above factors into consideration, we propose a **R**esource **A**llocation model based on **D**eep **Re**inforcement Learning and Generative **A**dversarial **N**etworks (RADEAN). We apply GAN to predict the future resource utilization of edge servers, and make resource allocation decisions through a multi-replay memory DRL agent with priority to maximize the long-term average resource utilization. Meanwhile, we guarantee the transmission latency and execution time of tasks to ensure the users' QoS. The contributions are as follows:

- We propose a RADEAN model for efficient resource allocation, which uses an improved Deep Q Network (DQN) with multiple replay buffers to reduce the interaction between multiple samples, and prioritize samples to further promote the decision performance of the proposed model.
- We apply GAN to forecast the CPU resource usage of the next episode, exploiting the semi-supervised learning to enhance prediction accuracy and considering it as one of bases for better resource allocation decisions.
- We conduct experiments on real-data sets, validating that RADEAN exceeds traditional and advanced resource allocation methods with great generality while ensuring transmission latency and execution time. The ablation experiments prove the effectiveness of multi-replay memory and priority settings.

2 Related Work

Resource allocation refers to assigning the task sets with different QoS requirements to densely distributed edge/fog nodes, which is determined to achieve the goals of improved resource utilization, decreased energy consumption, makespan, and cost, or lower task calculation latency [14]. We present several methods for resource allocation in edge computing.

Traditional: LF Bittencourt et al. [15] used three different policies to conduct resource management based on application characteristics. J Lin [16] introduced Lyapunov optimization to propose a dynamic resource scheme to minimize cost and maximize system utility. A hybrid market-based resource transaction mechanism was established in [17] to maximize utility. But these methods were complex with high computational cost and unable to obtain optimal solutions.

Heuristic and Meta-heuristic: Studies applied a multi-criteria-based policy [18] and a heuristic-based [10] resource allocation scheme. Meta-heuristic approaches dealt with various constraints caused by heuristic methods. Some work suggested game-theoretic resource management techniques to minimize infrastructure energy consumption and costs [3]. While the other developed algorithms to determine the Stackelberg equilibrium and chose the best resource demand strategy [11]. Such mechanisms were prone to fall into local minima.

Deep and Reinforcement Learning: Authors utilized an LSTM-based algorithm [5] and multiple parallel deep neural networks [19] for resource management in edge. Reinforcement learning (RL) had stronger self-improvement ability than deep learning. Researchers exploited the actor-critic reinforcement learning [20] and a parameterized DQN [6] to solve the joint decision-making issue, minimizing the average end-to-end latency. A DRL-based resource allocation scheme with minimum computing cost was created in [4]. None of them studied the impact of future resource conditions on allocation decisions.

Hybrid: Several hybrid approaches combing other algorithms with DRL were employed for resource allocation. Heuristic and meta-heuristic algorithms were incorporated with RL in [21] and [22], respectively. And in [23], a recurrent neural network-based long short-term memory combined with DRL was completed. We also apply a hybrid method, which uses an improved DQN and GAN to achieve the maximum long-term average resource utilization.

3 Problem Definition

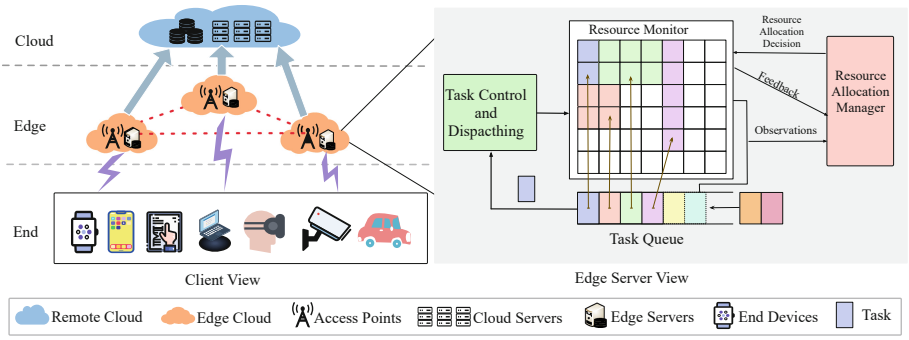


Fig. 1. The scenario of resource allocation in edge computing.

As shown in Fig. 1, it is a typical cloud-edge-end three-layer architecture. In intelligent driving application scenario, various smart devices such as self-driving cars, continuously offload tasks to the edge layer for certain computing resource requests from edge servers. The edge layer includes multiple edge clouds, which

is usually equipped with multiple edge servers. The Task Queue in the edge server stores tasks from terminal devices and their execution locations are determined by Task Control and Dispatching agent. If the edge server cannot meet the resource requirements of the tasks currently, tasks will be migrated to another edge server or forwarded to the remote cloud for procession. The Resource Monitor observes the edge server's resource usage. Based on the resource utilization status, the Resource Allocation Manager runs the resource allocation strategy to assign its own resources to all tasks on this server to meet user demands. The edge clouds cooperate with the remote cloud, which handles the tasks that can not be processed due to computation limitation of the edge layer. Also, the remote cloud defines and adjusts AI reasoning model, being responsible for unified operation management, manual review and persistent storage of key data. It makes the entire system more efficient and supportable for richer applications.

Unreasonable resource allocation mechanisms affect users' QoS and interests of service providers. If a task is allocated too few computing resources, its execution time will increase. And there will be a congestion in the task queue due to numerous unfinished tasks, which greatly raises the overall latency of tasks. Besides, the resources that could have been used will be idle instead, reducing the resource utilization of the edge server. If too many computing resources are allocated to a task, it may lead to resource deficit and data migration for other tasks. The subsequent increased data migration costs and transmission delay greatly influence latency-sensitive tasks especially. Higher latency will cause tasks such as target detection not be able to get feedback in time, which is detrimental both to users and service providers.

Assume that within a period of time, a total of V jobs are received from the view of the edge server. The job set is expressed as $\mathcal{J} \triangleq \{j_1, j_2, \ldots, j_V\}$. Each job $j_v, j_v \in \mathcal{J}$ contains multiple tasks and each task includes multiple instances. There are N tasks in all and they are stored in the Task Queue $\mathcal{Q} \triangleq \{q_1, q_2, \ldots, q_N\}$, according to their arrival time and following the First In First Out (FIFO) principle. For each task $q_n, q_n \in \mathcal{Q}$, the profile of each task is $q_n = (c_n, \omega_n, \rho_n, \xi_n)$, where c_n means the amount of requested computing resources, ω_n and ρ_n are the number of instances and data volume carried by q_n, respectively. The symbol ξ_n represents the deadline request of task q_n.

For simplicity, we only consider computing resources here. Set the computing resource capacity of an edge server as K, and the length of the sliding time window as H. The Resource Monitor observes the resource usage and pays close attention to a matrix $\mathcal{M}$ with size $K \times H$. Each row of $\mathcal{M}$ represents the state of a certain computing resource from current time to H timesteps in the future, $m(k,:) = \{m_{k,1}, m_{k,2}, \ldots, m_{k,h}, \ldots, m_{k,H}\}, k \in \{1, 2, \ldots, K\}$. Each column of $\mathcal{M}$ depicts the status of all computing resources at the current timestep, $m(:,h) = \{m_{1,h}, m_{2,h}, \ldots, m_{k,h}, \ldots, m_{K,h}\}, h \in \{1, 2, \ldots, H\}$. Each unit $m_{k,h} \in \{0, 1\}$ indicates the usage of the computing unit of the k-th row at h-th timestep, where 1 means that the resource is already occupied, while 0 reveals not. $\mathcal{M}$ is a matrix of all 0 s initially, illustrating that all computing resources are available.

Table 1. Table of notations and representations

Type	Nota.	Representations
Job	$\mathcal{J}$	the set of jobs
	j_v	the v-th job in job set $\mathcal{J}$
	V	the total number of jobs
Task	q_n	the n-th task
	c_n	the amount of requested computing resources of task q_n
	ω_n	the number of instances carried by task q_n
	ρ_n	the data volume carried by task q_n
	ξ_n	the deadline request of task q_n
	l_n	the time offset of task q_n relative to the sliding window
	d_n	the execution duration time of task q_n
	N	the total number of tasks
Edge Server	K	the quantity of computing resource capacity
	$\mathcal{M}$	the observation resource matrix of the edge server
GAN	y_e	the real resource utilization at episode e
	$\tilde{y}_{e+1}$	the predicted resource utilization at episode $e+1$
	F	the length of prediction series
DRL	$\mathcal{Q}$	the task queue
	$\mathcal{Q}_A$	the observation task sub-queue
	L	the length of task queue $\mathcal{Q}$
	H	the length of sliding window
	a_n	the action (resource allocation decision) of task q_n
	s	the current state of the edge server
	r_n	the reward after performing action a_n
	s'	the next state after performing a_n

At timestep t, we make a resource allocation decision a_n for each task q_n, determining how many resources it will be allocated per unit timestep. Its execution duration time d_n can be calculated as $d_n = c_n/a_n$. Next, under the least time offset, we search for an all 0 s sub-matrix with size $a_n \times d_n$ in the current sliding window and set all values in the sub-matrix to 1, indicating that the corresponding resources are allocated to task q_n. After that, the resource utilization of the edge server will be updated to u_t accordingly. The sliding window moves forward by a timestep. The specific length of each time unit can be determined in terms of actual application conditions. As shown in (1), given a certain period of time including T timesteps, our goal is to maximize the average resource utilization of the edge server in a long-term view, while ensuring the completion time of tasks for users' QoS. The sum of the resource requests of all tasks on an edge server cannot exceed its resource capacity limitation.

$$\Pi(T) = \arg\max \frac{1}{T} \sum_{t=1}^{T} u_t, \quad \text{s.t.} \sum c < K. \tag{1}$$

Relevant notations and representations are shown in Table 1. A few points to note: First, the task requests submitted by users are completely offloaded from end devices to the edge server for execution, including all the carried data. All instances carried by the task execute in parallel. Second, we suppose that once the task starts, it will not be interrupted due to any reason, such as the termination of hardware factors, forced surrender of resources for higher-priority tasks, power failure, network connection interruption and so on. Third, the execution locations of tasks that have already begun are not allowed to change, nor are they allowed to migrate between multiple edge clouds or multiple edge servers.

4 Proposed Model

In this section, we first introduce the workflow of the proposed RADEAN, and then present the GAN-based computing resource prediction module and DRL-based resource allocation module.

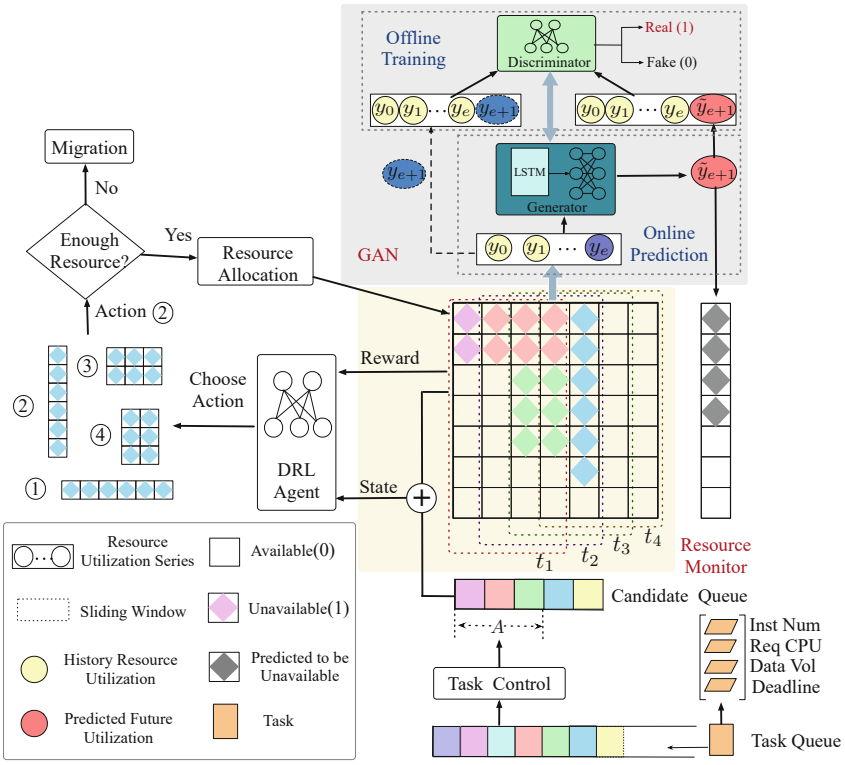


Fig. 2. The framework of RADEAN

The workflow of RADEAN is shown as Fig. 2. Tasks are sorted in the Task Queue according their arriving time and performed following FIFO principle. The tasks sequentially passed through the Task Control module, which is responsible for judging whether the task meets the basic conditions for execution on this edge server. If the deadline requirement of the task cannot been met under infinite resources, the task will be rejected. Otherwise, the task enters the Candidate Queue. The Resource Monitor supervises the server's resource usage. We select a sub-queue $\mathcal{Q}_A$ from the Candidate Queue, the size of which is A. The DRL-based agent decides the amount of allocated computing resources per time unit to the task and its start time, according to the current resource status, task sub-queue state and future resource utilization predicted by GAN. Then, the edge server checks whether the resource allocation requirement can be satisfied in the current sliding widow (sufficient computing resources and qualified all-zero sub-matrix to avoid prolonged waiting time). If so, the task is processed immediately. Otherwise, the task is migrated for execution.

When task is assigned relevant computing resources, the resource condition of the edge server modifies accordingly. The Resource Monitor calculates the resource utilization of the current sliding window, and feeds it back to the agent as a reward for resource allocation strategy adjustment. Then the sliding window moves forward one timestep, preparing for the next task's resource allocation.

We decouple the time into two phases, episode and timestep. There are E episodes in total and one episode contains A multiple timesteps. Resource allocation is executed for a task at each timestep. GAN predicts the future resource occupation $\tilde{y}_{e+1}$ of the next episode $e+1$, which is performed only after an episode ends. The GAN is trained in advance and only the Generator is used to conduct the online resource utilization prediction when making resource allocation decisions. The resource utilization records of the sliding window at different moments form a time series. And the Generator is trained offline and regularly updated through the Discriminator and these new data.

Tasks in edge are dynamic and heterogeneous. They are submitted from various applications with different resource requests and optimization subjects, arriving in arbitrary time and order. This makes the future resource occupancy has great uncertainty, which in turn affects the resource allocation decision and execution of tasks, influencing users' QoS. Consequently, it is meaningful to predict the future computing resource utilization. GAN is suitable and qualified for resource prediction: Firstly, for characterizing resource workload, one of the challenges of an effective fitting process is the prior selection of the model type, such as normal distribution [12]. While GAN does not require prior knowledge or assumptions about the sample distribution and underlying models themselves. It performs effective model descriptions based on the sample data, and estimate appropriate parameters. Secondly, GAN mines the information behind the resource utilization data, which is far richer than that expressed by sparse labels. It generates clearer and more real data according to the characteristics of the historical resource utilization trace of edge server and is superior to traditional neural network classification and feature extraction. Finally, the production cost

of the sample set is very high, while GAN can perform semi-supervised learning based on a small amount of trace data instead of massive data to train networks and complex models. The adversarial training between the Generator and the Discriminator further improves the prediction effect of the Generator.

GAN provides the more accurate predicted resource utilization of edge servers, contributing a more valuable state input for DRL to enhance resource allocation strategies and improve the overall system performance. DRL does not require manually labeled training samples and complication operations to obtain optimization results. It acquires resource utilization data from the Resource Monitor for training and adjusts the strategy based on feedback. For achieving the long-term maximum average resource utilization, DRL is more robust and adaptable to dynamic edge environment, benefiting both users and service providers.

Although the replay memory setting solves the problem of increased computational complexity and storage cost of Q-learning due to exponential number of states and actions, there are mutual influences between samples [8]. Therefore, we propose an improved DQN model, which puts transition samples into different replay buffers with various parameters according to the predicted resource occupancy values to achieve better exploration. It saves the space to store training data, ensures the diversity of samples, and eliminates the mutual sample influences to enhance data efficiency and training effect. Moreover, transitions may be more or less surprising, redundant than expected, and may not even be immediately useful to the agent. The key to the effectiveness of relevant transitions lies in the mass of highly redundant failure [24]. So it is not efficient enough to sample uniformly in replay buffers. We apply prioritized sampling to further improve the utilization of samples, reduce the number of training iterations, and promote the performance of the decision model.

4.1 Resource Prediction Based on GAN

GAN is pre-trained and its parameters are updated regularly using the historical resource trace. It includes a Generator and a Discriminator. In online prediction at episode e, the Generator predicts the resource utilization percent of episode $e+1$, using the historical trace series of previous F moments, $y_r = \{y_{e-F+1}, \ldots, y_{e-1}, y_e\}$. In offline training, the predicted result $\tilde{y}_{e+1}$ is combined with the real sequence y_r to form $y_G = \{y_{e-F+1}, \ldots, y_{e-1}, y_e, \tilde{y}_{e+1}\}$. There is another real trace $y_{G'} = \{y_{e-F+1}, \ldots, y_{e-1}, y_e, y_{e+1}\}$, in which y_{e+1} is the real resource utilization of episode $e+1$. The two series y_G and $y_{G'}$ are input into the Discriminator to be distinguished. The networks continue training until the Discriminator cannot recognize the real data and generated ones. At this time, the Generator can obtain the data distribution and accurately predict the computing resource occupancy. Here, the Discriminator differentiates the entire sequence rather than the single resource utilization at episode $e+1$, y_{e+1} and $\tilde{y}_{e+1}$. This design better captures the correlation between y_{e+1} and the sequence information [13]. As (2) shows, the output of Long Short-Term Memory (LSTM) is input into a fully connected layer to obtain the prediction result in Generator,

$$G(y_r) = \tilde{y}_{e+1} = \delta(Wg(y_r) + b), \tag{2}$$

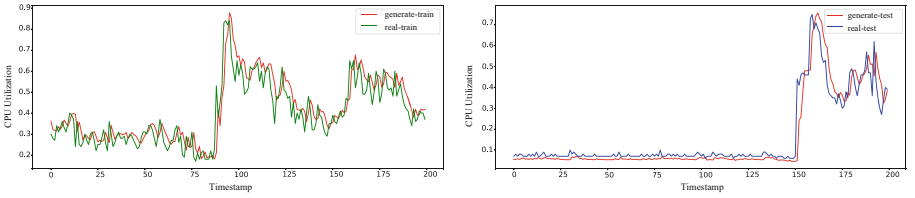
where $g(\cdot)$ is the output of LSTM, δ means the ReLU activation function, W and b are the weights and bias in the fully connected layer, respectively. A Multi Layer Perceptron (MLP) including three fully connected layers is regarded as the Discriminator and Leaky ReLU is applied as the activation function between hidden layers. As shown in (3), Sigmoid function is used in the output layer,

$$D(y_G) = \sigma(f(y_G)), D(y_{G'}) = \sigma(f(y_{G'})), \tag{3}$$

where $f(\cdot)$ is the output of connected layers and σ is activation function. Both networks are trained using Binary Cross Entropy Loss and the value function in (4) shows the difference between real resource utilization and generated ones,

$$\min_G \max_D V(G, D) = E(\log D(y_{G'})) + E[\log(1 - D(y_G))]. \tag{4}$$

Compared to the simple use of LSTM, this adversarial training method improves the accuracy of prediction. We set $F = 4$, using the resource occupancy rate of the previous 4 episodes to predict the next one. The real usage trace of 3 machines is chosen randomly from Cluster-trace-v2018[1], with a total of 64946 sequences. It is divided to a training and a test set according to the ratio of 8:2. The number of neurons in the hidden layer of Generator is 512. The initial historical resource utilization of GAN is randomly generated. As shown in Fig. 3, we select 200 prediction results in the training and test sets. It can be seen that GAN can accurately predict the resource utilization at the next moment.



(a) The results on training data (b) The results on test data

Fig. 3. Resource utilization prediction results by GAN

4.2 Resource Allocation Based on DRL

In our design, the DRL agent acts as the Resource Allocation Manager, and the Resource Monitor is treated as the environment. Based on the observed states of resource usage and task queue in the edge server, the agent makes resource allocation actions, determining the amount of allocated resources for each task. Then the Resource Monitor modifies the resource condition and feeds the reward back to the agent to adjust the resource allocation strategy accordingly.

[1] https://github.com/alibaba/clusterdata/.

State: The current state s is a triplet $s = (\mathcal{M}, \mathcal{Y}, \mathcal{Q}_A)$, where $\mathcal{M}$ represents the current computing resource matrix of the edge server, $\mathcal{Y}$ describes the predicted computing resource matrix of the next episode, and $\mathcal{Q}_A$ means the observation of the task sub-queue. The size of $\mathcal{M}$ is $K \times H$ and the size of $\mathcal{Y}$ is $K \times 1$. The output computing resource utilization of the GAN is $\tilde{y}_{e+1} \in [0, 1]$. We mark the values of the first $\tilde{y}_{e+1} \times K$ units as 1 to indicate the predicted resource occupancy condition at the next episode. The original dimension of $\mathcal{Q}_A$ is $A \times 4$. Each task contains four features including the number of instances, the carried data volume, the amount of requested computing resources and the deadline limitation. In order to unify the dimensions, we transpose $\mathcal{Q}_A$ and expand its dimension to $K \times 4$. Therefore, the dimension of state s is $K \times (H + 1 + 4)$.

Action: The action a indicates the amount of computing resources allocated to the task q_n per time unit. We filter the action values output by the network to ensure its effectiveness [8]. The value of action a_n must be a divisor of the requested resource c_n, that is $c_n(\mathrm{mod} a_n) = 0$. When using the ε-greedy algorithm, the action a_n randomly selects a value that be divisible by c_n. If the optimal value output by the network is not exactly divisible, we select the divisor closest to the output as the action. The resource occupation time d_n can be calculated correspondingly and the start time l_n is the least time offset under the condition that the available resource sub-matrix for allocation is all 0 s.

Reward: Our goal is to maximize the average resource utilization in a time period from a long-term perspective. Hence, after the task is allocated resources at time t, we use the current resource utilization within the sliding window of the edge server to represent the reward $r = u_t$.

The improved DQN includes three replay buffers with priority. We conduct resource utilization prediction at each episode instead of each timestep, avoiding frequent prediction of GAN, and update the sample priority only in current buffer, decreasing the time complexity of the algorithm. At timestep t, based on the current state s, task q_n is assigned resources according to action $a = a_n$, acquiring the next state s'. We store the transition (s', a, r, s) in these three buffers according to the value of predicted resource utilization $\tilde{y}_{e+1}$. Different values of ε is applied for the ε-greedy algorithm in various buffers. Besides, we adopt TD-error as the sampling priority to enhance the original uniform sampling method, which represents the difference between the action value output by the current value function and the estimated ones. Higher TD-error indicates that the output of the current value function is less accurate, implying that there is still more information can be learned and larger space for improvement in the prediction accuracy from this sample. So the sample will be assigned higher priority. Data structure SumTree is used to implement proportional prioritization, whose complexity is only $O(\log n)$ [24]. To ensure that every stored transition can be sampled, the new transition will be given a large priority.

Algorithm 1. Algorithm of RADEAN

1: Initialize Network Q with θ and Target Q-network Q' with $\theta' = \theta'$; Initialize replay buffers B_1, B_2, B_3 with same size ζ and different parameters ε_1, ε_2, ε_3; Mini-batch λ; Average reward R_{avg}, Reward sum R_{sum}; Average execution time P_{avg}; Execution time sum P_{sum}; Average transmission latency Z_{avg}; Transmission latency sum Z_{sum}; Timestep T; Task num N; Step ς; Edge server capacity K; Sliding window length H; Data transmission speed η; Task sub-queue length A;
2: Set initial values of GAN prediction windows
3: Put jobs into priority queue $\mathcal{Q}$ according to arriving time
4: **for** $e = 0$ to E **do**
5: obtain current task q_n at timestep t; N+=1
6: Task control according to ξ_n
7: get the predicted resource utilization $\tilde{y}_{e+1}$ from GAN
8: Choose replay buffer B_i according to $\tilde{y}_{e+1}$
9: Form current state $s = (\mathcal{M}, \mathcal{Y}, \mathcal{Q}_A)$
10: With probability ε_i choose a_n randomly
11: Otherwise $a_n = \arg\max Q(s, a; \theta)$
12: Filter and choose valid a_n
13: Get duration time $d_n = c_n / a_n$
14: **if** no enough resource in H for q_n **then**
15: Migrate q_n; $Z_{\text{sum}} += 2 \times (\rho_n / \eta)$
16: **else**
17: T+=1; Perform action $a = a_n$; Get reward $r = u_t$
18: Gain updated resource matrix $\mathcal{M}'$
19: $R_{\text{sum}} += r$; $P_{\text{sum}} += d_n$; $Z_{\text{sum}} += (\rho_n / \eta)$
20: Sliding window forwards one timestep
21: Form new state $s' = (\mathcal{M}', \mathcal{Y}, \mathcal{Q}_A)$
22: Store Transitions (s', a, r, s) to B_i
23: Sample a mini-batch of transitions (s_{j+1}, a_j, r_j, s_j) from B_i
24:

$$\mathcal{Q}_{\text{Target}} = \begin{cases} r_j, & \text{for terminal } s_{j+1}, \\ r_j + \gamma \max\limits_{a'} Q'(s_{j+1}, a'; \theta'), & \text{otw.} \end{cases}$$

25: Compute TD-error $\varphi_j = \mathcal{Q}_{\text{Target}} - Q(s_j, a_j, \theta)$
26: Update transition priority $p_j \leftarrow |\varphi_j| + \epsilon$
27: Perform a gradient descent step on Loss value
28: **end if**
29: Every ς steps, update Q' using $\theta' = \theta$
30: **end for**
31: **return** $R_{\text{avg}} = R_{\text{sum}} / T$; $P_{\text{avg}} = P_{\text{sum}} / T$; $Z_{\text{avg}} = Z_{\text{sum}} / N$;

In the improved DQN, the Target Q-network Q' computes target Q-value $\mathcal{Q}_{\text{Target}} = r + \gamma \max\limits_{a'} Q'(s', a'; \theta')$, where γ indicates the impact of further moments on present. The loss is calculated as $\mathcal{L}(\theta) = \mathbb{E}[(\mathcal{Q}_{\text{Target}} - Q(s, a; \theta))^2]$. As shown in Algorithm 1, we first conduct task control and apply GAN to predict the future resource utilization (Lines 6–7). After selecting the replay buffer and obtaining current state, the ε-greedy algorithm is used to perform exploration and

exploitation (Lines 8–11). An action is randomly generated under probability ε_i in replay buffer B_i, otherwise the action with the maximum value function is selected. The value of ε_i keeps decreasing during the training process. Then we filter the valid values to determine action a (Line 12). When the resource allocation decision of task q_n is determined, we judge if there are enough resources in the observation window for q_n (Line 14). If not, we migrate the task to a nearest selected server in other edge clouds or the remote cloud, during which the transmission time of the task is at least twice that of executed on the local edge server (Line 15). Otherwise, the action is performed, the resources are allocated. The agent obtains the corresponding reward and the status of system is modified (Lines 17–21). The transition (s', a, r, s) are distributed to different buffers according to the predicted resource utilization $\tilde{y}_{e+1}$ (Line 22). Every time we select a mini-batch with size λ to update the network, which performs iterative update with reduced complexity (Line 23). TD-error is computed to update the transition priority, and ϵ is used to ensure the value of transition priority is non-zero (Lines 25–26). The parameters of the Q network are utilized to update the Target-Q network every ς steps, which eliminates the divergence and oscillations during update (Line 29). Finally we compute and return the average resource utilization, average execution time and average transmission latency (Line 31).

5 Experiments

5.1 Experiments Setup

We deployed the simulation environment on the machine (2 CPU with 1 cores, type: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20 GHz, 48 GB memory) installed with Linux 5.4.0 system. All models are implemented by Python 3.8.10. Because preprocessed cloud task records can be used for experiments in edge [25], we select dataset Cluster-trace-v2018 (See Footnote 1), containing task execution and server status trace of 4000 machines in 8 days. We randomly choose the attributes of one server as a reference for edge server property settings. A job in this dataset contains multiple tasks, and each task includes multiple instances. The scheduling unit is task in our experimental design and five different test data sets of 10 jobs (50 tasks), 50 jobs (178 tasks), 100 jobs (357 tasks), 200 jobs (773 tasks), and 500 jobs (1827 tasks) are tested. In order to verify the performance of the RADEAN model, the following methods are selected as baselines for comparison.

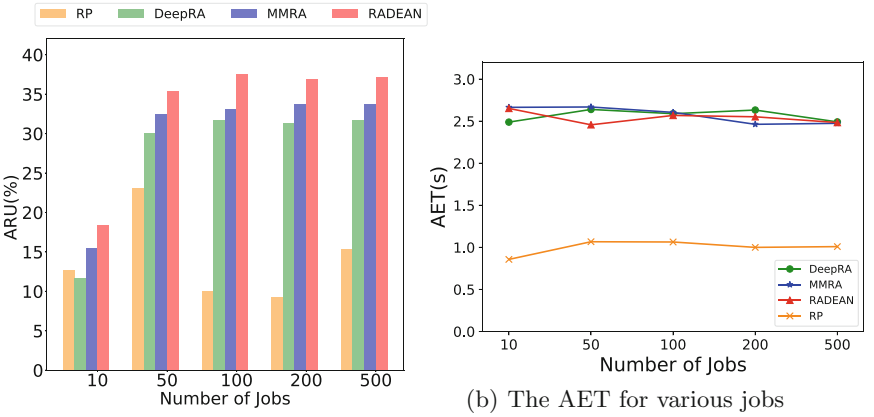
- Multi-Replay Memory-based Resource Allocation (MMRA): DRL with multiple replay buffers determines the resource allocation amount of tasks. The resource request of the next task is applied to calculate the resource utilization of the edge server at the next moment [8].
- Deep Reinforcement Learning-based Resource Allocation (DeepRA): It directly uses the ordinary DQN with a single replay buffer without priority for resource allocation and excludes resource prediction [9].

– Resource Precedence (RP): Within the scope of edge server computing resource capacity, it allocates the most available resources to meet the resource requests of tasks as many as possible [8].

5.2 Experiments Results and Analysis

Basic Experiments. Firstly, the computing resource capacity of the edge server is set to 48, half of the machine capacity in Cluster-trace-v2018 (See Footnote 1). Then we test the Average Resource Utilization (ARU) of different models to demonstrate the improved performance of RADEAN on resource utilization. At the same time, we present the Average Transmission Latency (ATL), the Average Execution Time (AET) and Migration Percent (MP) of tasks to measure the models' guarantee of QoS. All results are the average of five experimental operations.

It can be seen from Fig. 4(a), the ARU of RADEAN is highest for task sets of different scales. The effect of RP is the worst and the ARU of RADEAN is 4 times that of RP when job set is 200. Because RP allocates as many available resources as possible based on the task's request. At this time, the execution time of the task is the shortest, which confirms the result in Fig. 4(b) that the AET of RP is much lower than that of other models. The reason is that when the sliding window moves forward, the previous tasks may have been finished and will not occupy the resources in the current sliding window, gaining lower ARU. However, by using other three models, tasks will occupy several consecutive time units, thereby obtaining higher ARU and longer AET.



(a) The ARU for various jobs

(b) The AET for various jobs

Fig. 4. Experiment results when K is 48

DeepRA only makes resource decisions based on current resource status and task queue, ignoring the future resource utilization. RADEAN outperforms DeepRA by 58.08% on 10 job set and 17.08%–18.34% on others. Compared

with DeepRA, MMRA has better ARU performance, because MMRA not only considers future resource occupancy, but also uses multiple buffers to eliminate the interaction between experiences. But RADEAN still has a maximum performance improvement of 19.32% than MMRA when job set is 10. The reason is that the future resource utilization of MMRA is only obtained by the resource request of the next task in the task queue. It ignores the resource requests of subsequent tasks and the distribution of the resource utilization time series of the edge server, only accomplishing a short-term sub-optimal decision result.

RADEAN applies GAN to predict the resource utilization, considering previous a few timesteps of historical data comprehensively. It better mines the hidden information behind the training samples and captures the correlation between the resource utilization at the next moment and the previous time series. The adversarial learning between the Generator and Discriminator further improves the prediction accuracy. Moreover, RADEAN assigns different priorities to samples, which makes better use of rare and important experience information and achieves higher ARU in a long-term view.

As displayed in Fig. 4(b), apart from RP, RADEAN has the minimum AET on 50 jobs and has near AET with MMRA and DeepRA on other data sets. In Table 2, we see the ATL results of various models. RADEAN is closest to the optimal results of RP when job sets are 100 and 500, and achieves the best ATL on the rest. Hence, RADEAN attains excellent ARU with acceptable AET and ATL, ensuring the users' QoS and indicating its great generalization.

Table 2. The ATL and MP on various job sets

Cap.	Model	Job									
		ATL (s)					MP (%)				
		10	50	100	200	500	10	50	100	200	500
$K = 48$	RADEAN	**1.245**	**1.196**	1.669	**1.753**	1.849	**19.20**	**23.37**	62.96	**76.51**	85.90
	DeepRA	1.302	1.212	1.706	1.759	1.872	24.00	24.72	67.45	78.29	88.16
	MMRA	1.309	1.225	1.687	1.760	1.875	26.00	24.83	64.87	77.33	88.47
	RP	1.716	1.207	**1.554**	1.844	**1.823**	58.00	24.72	**52.10**	85.90	**83.14**
$K = 96$	RADEAN	1.196	1.048	1.390	1.521	1.784	14.40	7.98	36.25	53.68	79.29
	DeepRA	1.192	1.065	1.436	1.570	1.929	14.80	9.66	40.39	58.47	79.69
	MMRA	1.233	1.054	1.421	1.567	1.792	17.20	8.65	38.99	58.14	80.08
	RP	**1.061**	**0.965**	**1.202**	**0.993**	**1.767**	**2.00**	0.00	**18.21**	0.00	**77.39**

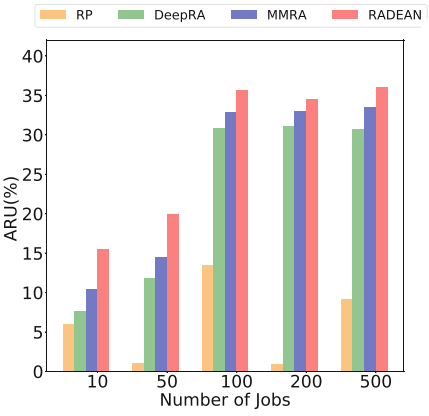
We also analyze the migration percentage of the tasks in Table 2. When K is 48, we see that the outcomes of RADEAN are near to the best results RP on job set 100 and 500, and RADEAN has the least migration amount in other cases. The increase in the task migration ratio leads to a growth in data transmission delay, which corresponds exactly to the ATL results in Table 2. RADEAN not only reduces ATL, ensuring users' QoS, but also decreases the migration of task data, cutting down the cost of task migration. We find that when the number of tasks is large enough, the resources of the edge server can no longer meet task requirements, which leads to a very high percent of task migration. Therefore, we

increase the resource capacity configuration of the edge server, and then proceed to verify the performance of the proposed model.

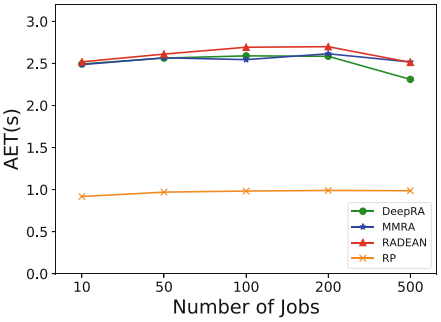
Table 3. New results on two new task sets

Task Num	ARU (%)	AET (s)	ATL (s)	MP (%)
178	5.31	0.966	1.024	0.56
773	13.07	0.998	1.213	21.34

Extended Experiments. We set the resource capacity of edge server to be 96, the same as the configuration of the selected machine in the data trace. In Fig. 5, it can be seen that the ARU of RADEAN is still higher than other baselines, achieving the same level of AET of tasks as other models except RP. Compared with MMRA, RADEAN has the largest performance improvement of 48.21% when job set is 10. Besides, when $K = 96$ in Table 2, RP has the least ATL. Because setting larger resource capacity satisfies the resource requests for more tasks when resource precedence mechanism is adopted, reducing task migration and ATL. RADEAN accomplishes the sub-optimal solution. Also, as Table 2 displays, the migration ratio of tasks decreases after we raise the edge server resource capacity. When the job numbers are 50 and 200, the ARUs of RP are extremely low in Fig. 5(a) and the migration percent of RP is 0 in Table 2. We analyze that this is because the resource requests of most tasks in these two data sets are far lower than the resource capacity of the edge server. Therefore, tasks are allocated with maximum resources and processed quickly, achieving very short execution duration time. There are few resources occupied by unfinished tasks in the sliding window, thereby gaining extremely small ARU. As Table 3 shows, we randomly sample another two new task sets with same size, and find the ARUs of RP increase, which verifies our explanation.



(a) The ARU for various jobs



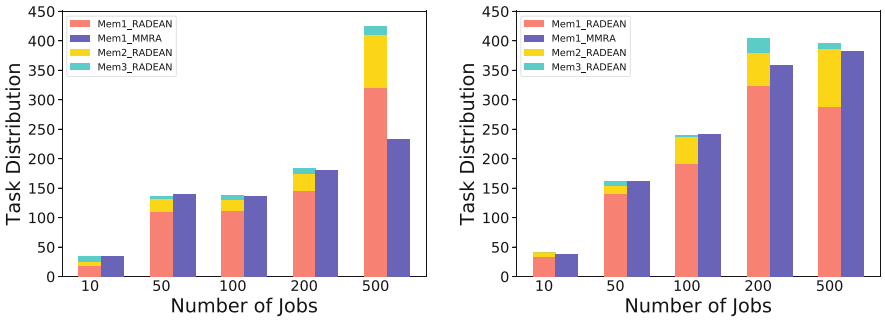
(b) The AET for various jobs

Fig. 5. Experiment results when K is 96

Besides, in Fig. 6, we display the sample distribution in various replay buffers when the numbers of migration in RADEAN and MMRA models are the least among five experimental results. The tasks performed by RADEAN are the same as or higher than MMRA's. Moreover, it can be seen that regardless of the capacity of computing resources, experiences will be stored in three reply buffers of RADEAN. While MMRA stores all samples in one of the three replay buffers in all cases. These results prove that the proposed model RADEAN distributes samples into various buffers more evenly compared with MMRA. And it better predicts the resource utilization of edge server from a long-term view, separates the experiences, and reduces the interaction between them.

Ablation Study. In order to verify the impact of multi-replay memory and sampling priority settings in RADEAN on resource allocation decisions, we conduct ablation experiments, which use GAN to predict the future resource utilization. The relevant models are described as follows:

- GM: Resource Allocation Model with GAN and Multiple Non-priority Replay Buffers (GM). It verifies the influence of the priority in replay buffers.
- GSP: Resource Allocation model with GAN and a Single Replay Buffer with Priority (GSP). It certifies the significance of multi-replay memory.
- GS: Resource Allocation Model with GAN and a Single Non-priority Replay Buffer (GS), proving the impact of both priority and multi-replay memory.

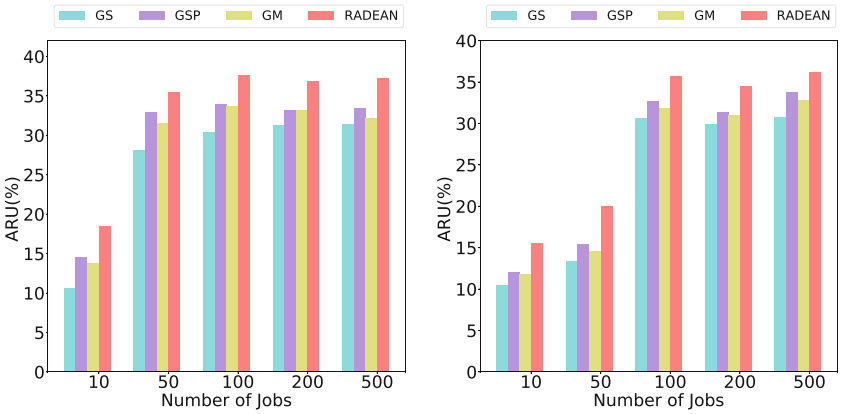


(a) The results when K=48 (b) The results when K=96

Fig. 6. Sample distribution on various cases

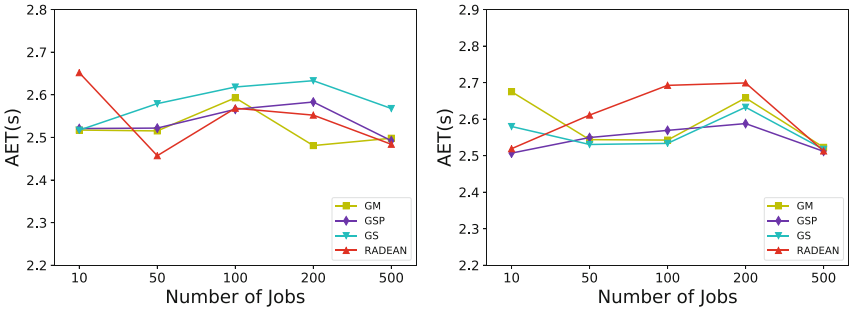
As shown in Fig. 7, GS has the worst effect and RADEAN has the best results. The maximum performance enhancement of RADEAN reaches 74.40% compared with GS when K=48 and job set is 10. This shows that setting multi-replay memory and adding priorities improve the performance of resource allocation. Multiple replay buffers reduce the mutual influence between samples, thereby enhancing the training effect. So RADEAN obtains higher ARU in the long-term view. By assigning different weights for samples, those experiences which

are more important and useful for resource allocation decisions have a greater chance of being sampled. This design further advances the model effect. The ARU of GSP is slightly higher than that of GM, indicating that priority has a higher enhancement than multi-replay memory. The performance improvements of GSP and GM are 4.55%–37.09% and 2.61%–30.26% compared with GS, respectively.



(a) The ARU result when K=48 (b) The ARU result when K=96

Fig. 7. The ARU ablation experiment results

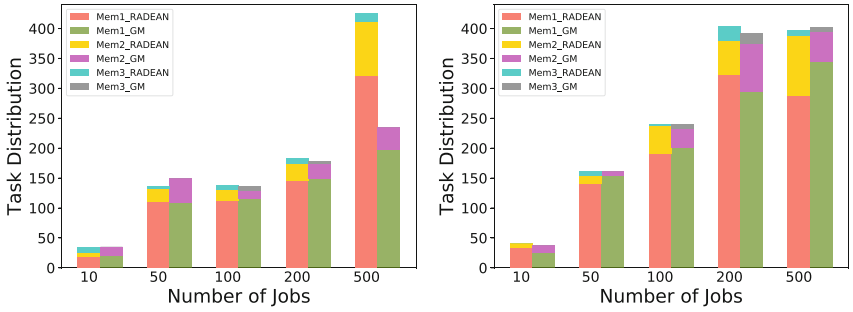


(a) The AET when K=48 (b) The AET when K=96

Fig. 8. The AET ablation experiment results

Table 4. The ATL and MP ablation results on various job sets

Cap.	Model	Job									
		ATL (s)					MP (%)				
		10	50	100	200	500	10	50	100	200	500
$K = 48$	RADEAN	**1.245**	**1.196**	**1.669**	**1.753**	**1.849**	**19.20**	**23.37**	**62.96**	**76.51**	**85.90**
	GM	1.277	1.209	1.687	1.763	1.866	22.00	23.82	64.76	77.64	87.56
	GSP	1.279	1.214	1.695	1.760	1.864	21.60	24.72	65.71	77.25	87.35
	GS	1.284	1.212	1.694	1.761	1.867	22.80	25.28	66.67	78.58	87.74
$K = 96$	RADEAN	**1.196**	**1.048**	**1.390**	1.521	1.784	**14.40**	**7.98**	**36.25**	**53.68**	**79.29**
	GM	1.215	1.067	1.441	**1.501**	**1.782**	16.00	10.45	40.62	54.13	80.14
	GSP	1.216	1.054	1.404	1.536	1.789	16.00	9.89	37.31	55.16	79.83
	GS	1.226	1.077	1.450	1.546	1.792	16.80	10.90	41.68	56.22	80.14



(a) The results when K=48 (b) The results when K

Fig. 9. Sample distribution ablation results

In Fig. 8, the AET of RADEAN is close to or slightly higher than other ablation models. But a higher ARU is realized with the acceptable time increase. From Table 4, we can see that RADEAN gets the smallest migration percent of tasks in all conditions and GS has the worst results. Also, RADEAN has the optimal ATL results almost in all conditions, and GS has the highest ATL. This is because GS lacks a multi-replay memory design, which leads to mutual influence between samples. The uniform sampling method of GS further reduces the effect of resource allocation decisions, leading to an increase in the MP of tasks and ATL. We also count the number of samples in different replay buffers when GM and RADEAN migrate the least number of tasks among the five experiments. As Fig. 9 displays, RADEAN distributes samples more evenly to different buffers compared with GM. What's more, RADEAN executes more tasks, which reduces the MP and migration cost of tasks.

6 Conclusion

Performing resource allocation efficiently of the edge server is of great significance both to users and service providers. In this paper, we have proposed a model

named RADEAN, which uses an improved deep reinforcement learning of multiple replay buffers with priority for resource allocation. Combined the resource utilization rate predicted by GAN, current resource usage and task queue status together, DRL makes resource allocation decisions for tasks. We have conducted experiments on real-word data. The results prove that RADEAN obtains higher average resource utilization with generalization than traditional and state-of-the-art methods under acceptable task execution time and transmission latency, reaching the maximum performance improvement of 48.21% over MMRA. The ablation experiments confirm the effectiveness of the multi-replay memory and priority settings to improve the resource allocation performance.

References

1. Alnoman, A., Sharma, S.K., Ejaz, W., et al.: Emerging edge computing technologies for distributed IoT systems. IEEE Netw. **33**(6), 140–147 (2019)
2. Anu, A.S.: Resource allocation in fog computing based on meta-heuristic approaches: a systematic review. IJCSNS **22**(9), 503 (2022)
3. Zakarya, M., Gillam, L., Ali, H., et al.: epcAware: a game-based, energy, performance and cost-efficient resource management technique for multi-access edge computing. IEEE Trans. Serv. Comput. **15**(3), 1634–1648 (2020)
4. Zhang, Y., Zhang, M., Fan, C., et al.: Computing resource allocation scheme of IOV using deep reinforcement learning in edge computing environment. EURASIP J. Adv. Sig. Process. **2021**(1), 33 (2021)
5. Dlamini, T., Vilakati, S.: LSTM-based traffic load balancing and resource allocation for an edge system. Wireless Commun. Mob. Comput. **2020**, 1–15 (2020)
6. Liu, T., Ni, S., Li, X., et al.: Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing. IEEE Trans. Mob. Comput. **22**(7), 3870–3881 (2023)
7. Sharif, Z., Jung, L.T., Razzak, I., et al.: Adaptive and priority-based resource allocation for efficient resources utilization in mobile edge computing. IEEE Internet Things J. **10**(4), 3079–3093 (2021)
8. Xiong, X., Zheng, K., Lei, L., et al.: Resource allocation based on deep reinforcement learning in IoT edge computing. IEEE J. Sel. Areas Commun. **38**(6), 1133–1146 (2020)
9. Lakhan, A., Mohammed, M.A., Obaid, O.I., et al.: Efficient deep-reinforcement learning aware resource allocation in SDN-enabled fog paradigm. Autom. Softw. Eng. **29**, 1–25 (2022)
10. Bhajantri, L.B., Gangadharaiah, S.: Heuristic-based resource allocation for Internet of Things in gateway centric multi-layer fog computing. In: Tuba, M., Akashe, S., Joshi, A. (eds.) ICT4SD 2022. LNNS, vol. 516, pp. 567–579. Springer, Singapore (2022). https://doi.org/10.1007/978-981-19-5221-0_54
11. Chen, Y., Li, Z., Yang, B., et al.: A Stackelberg game approach to multiple resources allocation and pricing in mobile edge computing. Future Gener. Comput. Syst. **108**, 273–287 (2020)
12. Haverkort, B.R., Finkbeiner, F., de Boer, P.T.: Machine learning data center workloads using generative adversarial networks. ACM SIGMETRICS Perform. Eval. Rev. **48**(2), 21–23 (2020)
13. Zhang, K., Zhong, G., Dong, J., et al.: Stock market prediction based on generative adversarial network. Procedia Comput. Sci. **147**, 400–406 (2019)

14. Jamil, B., Ijaz, H., Shojafar, M., et al.: Resource allocation and task scheduling in fog computing and internet of everything environments: a taxonomy, review, and future directions. ACM Comput. Surv. (CSUR) **54**(11s), 1–38 (2022)
15. Bittencourt, L.F., Diaz-Montes, J., Buyya, R., et al.: Mobility-aware application scheduling in fog computing. IEEE Cloud Comput. **4**(2), 26–35 (2017)
16. Lin, J., Huang, L., Zhang, H., et al.: A novel Lyapunov based dynamic resource allocation for UAVs-assisted edge computing. Comput. Netw. **205**, 108710 (2022)
17. Huang, X., Gong, S., Yang, J., et al.: Hybrid market-based resources allocation in mobile edge computing systems under stochastic information. Future Gener. Comput. Syst. **127**, 80–91 (2022)
18. Naha, R.K., Garg, S.: Multi-criteria-based dynamic user behaviour-aware resource allocation in fog computing. ACM Trans. Internet Things **2**(1), 1–31 (2021)
19. Wang, Z., Lv, T., Chang, Z.: Computation offloading and resource allocation based on distributed deep learning and software defined mobile edge computing. Comput. Netw. **205**, 108732 (2022)
20. Wei, Y., Yu, F.R., Song, M., et al.: Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor-critic deep reinforcement learning. IEEE Internet Things J. **6**(2), 2061–2073 (2018)
21. Lee, S., Lee, S.K.: Resource allocation for vehicular fog computing using reinforcement learning combined with heuristic information. IEEE Internet Things J. **7**(10), 10450–10464 (2020)
22. Udayakumar, K., Ramamoorthy, S.: Intelligent resource allocation in industrial IoT using reinforcement learning with hybrid meta-heuristic algorithm. Cybern. Syst. **54**(8), 1241–1266 (2022)
23. Math, S., Tam, P., Kim, D.Y., et al.: Intelligent offloading decision and resource allocations schemes based on RNN/DQN for reliability assurance in software-defined massive machine-type communications. Secur. Commun. Netw. **2022**, 4289216 (2022)
24. Schaul, T., Quan, J., Antonoglou, I., et al.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015)
25. Liu, L., Tan, H., Jiang, S.H.C., et al.: Dependent task placement and scheduling with function configuration in edge computing. In: Proceedings of the International Symposium on Quality of Service, pp. 1–10 (2019)