Multi-agent Cooperative Computing Resource Scheduling Algorithm for Periodic Task Scenarios

Zheng Chen[1], Ruijin Wang[1](✉), Zhiyang Zhang[1], Ting Chen[1](✉), Xikai Pei[1,2], and Zhenya Wu[2]

[1] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 611731, Sichuan, China
{ruijinwang,brokendragon}@uestc.edu.cn, peixikai@cdatc.com

[2] The Second Research Institute of Civil Aviation Administration of China, Chengdu 610041, Sichuan, China
wuzhenya@cdatc.com

Abstract. The scheduling of large-scale service requests and jobs usually requires the service cluster to fully use node computing resources. However, due to the increasing number of server devices, the dependence between resource allocation and request, and the periodic external request received, the scheduling process of edge-oriented service requests is a complicated scientific problem. Existing studies do not take into account the periodic characteristics of service requests in different periods, leading to inaccurate scheduling decisions on external requests. This paper proposes a coordinated Multi-Agent recurrent Actor-Critic, based on a recursive network. CMARAC is used to solve the problem of computing resource allocation for periodic requests in edge computing scenarios. According to different resource information in the server cluster and the status of the task queue, the system state information and historical information are captured and maintained by integrating LSTM, and then the most appropriate service resources are selected by processing them in the Actor-Critic network. Tracking experiments using actual request data show that CMARAC can successfully learn the periodic state between external requests in the face of large-scale service requests. Compared with the baseline, the average throughput rate of the system implemented by CMARAC is improved by 2.1%, and the algorithm convergence rate is improved by 0.69 times. Finally, we optimized the parameters through experiments and determined the best parameter configuration of CMARAC.

Keywords: Service request · Multi-agent cooperative scheduling algorithm · Edge computing · Task scheduling

1 Introduction

The popularity of Chat GPT shows that AI is moving towards "big models, big data", and often has limited hardware resources to support the training of large

C. Li et al. (Eds.): APPT 2023, LNCS 14103, pp. 76–97, 2024.
https://doi.org/10.1007/978-981-99-7872-4_5

models. In the traditional cloud computing mode, there is a large delay in data transmission [1], so to realize the training and application of large-scale models, edge computing technology has become a field of concern. Edge computing is increasingly popular in the existing academic and industrial computing frameworks [2,3], and users are more inclined to share large hardware platforms [4,5], so the dynamic allocation and balance of limited hardware resources is the main problem facing edge computing.

The dynamic workload of the edge cluster environment brings a high challenge to the service request of the front-end application. When the request arrives, how to optimize the allocation of computing resources between the edge node and the cloud is a complex scientific problem [6].

In the existing work, there are a variety of solutions for task unloading and resource optimal allocation, such as meta-heuristic algorithms [7], dynamic programming [8,9], reinforcement learning [6,10], which can solve the problem of resource allocation to a certain extent. However, it is difficult to capture the periodicity of the task when completing the periodic task scheduling in the above scheme, so the optimal scheduling effect cannot be achieved. Using historical data to predict the future often means potential advantages [11]. Therefore, in marginal scenarios, how considering the periodic relationship of service requests in time and adjusting the plan of resource allocation is a key issue in resource allocation and request scheduling.

Aiming at the above problems, a method of coordinated Multi-Agent recurrent Actor-Critic (CMARAC) based on the recursive network is proposed in this paper. First, the reward function and network updating scheme of reinforcement learning are defined. Secondly, the history and reality of the server cluster are combined into a matrix with temporal characteristics and input into the network model to obtain the node selection probability vector. Finally, the node most suitable for the current request is selected according to the probability vector, and the state changes of the system after selecting this node are calculated and applied to the network input in the next period.

The specific scheme of this paper is as follows: (*i*) The introduction of a time series network in reinforcement learning can learn long-term strategy information and improve the performance of agents. (*ii*) The parameters and network structure of the temporal network are designed, and the reward function of reinforcement learning is defined to ensure the convergence of the network. (*iii*) Apply the proposed model to the experimental process of specific data sets to prove the effectiveness and superiority of the model. Therefore, the major challenges facing this scheme are arranging the time scale of the recursive network, how to design the reward function that can make the network converge, and finally proving the validity of the model.

In response to the above challenges, this paper adopts the method of dual time scale for resource scheduling and recursive network input design and sets the product of throughput rate and normalized negative exponential function of resource consumption variance within a period as the reward function. Finally, the validity of the model is proved by mathematical proof and experimental proof.

Based on the above work, the main contributions are as follows.

1. A multi-agent reinforcement learning method based on the recursive network is proposed. According to the distribution of requests in time and the deployment of servers in edge scenarios, a multi-agent reinforcement learning method based on the recursive network is designed. Based on the recursive network method, the input design under double time scales is adopted. The model can grasp the distribution characteristics of requests in time and output a more appropriate resource scheduling scheme. Multi-agent reinforcement learning can handle dynamic request information and cope with distributed time-varying resource systems.
2. A reward function is designed to consider both the system throughput rate and load balance within a period. Different from calculating the reward function once in each resource allocation process, this paper takes the negative exponential function value of the system throughput rate in multiple time units as a partial reward. At the same time, the normalized result of the variance of resource utilization ratio of different nodes is used as the index of cluster load balancing, and the negative exponential function value is also used as another part of the reward, and the product of the two parts is the complete reward function.
3. The experiment of the Alibaba cluster tracking data set is completed, which has obvious advantages compared with other scheduling algorithms. The parameters of the neural network structure are optimized and the optimal parameters are found through experiments. Compared with the algorithm without recursive network, the experiment shows that the convergence speed of CMARAC is faster. Finally, the feasibility of the algorithm in the real scheduling scenario is tested.

2 Related Research

Recent studies on similar resource scheduling problems have focused on reinforcement learning algorithms. In 2020, Chen et al. studied the joint power control of MECs in the Industrial Internet of Things and the dynamic resource management of computing resource allocation. To minimize the long-term average delay of the task, the original problem is transformed into a Markov decision process (MDP). A dynamic resource management (DDRM) algorithm based on deep reinforcement learning is proposed to solve the MDP problem.

In summary, the existing solution uses a single strategy for the scheduling process. It can not adapt to the complex and changing workshop production scene, with long order completion times, large system computational complexity, and poor stability of the scheduling effect achieved for different orders. This paper proposes an innovative RGA algorithm based on learning and meta-inspiration, which can adapt to the intelligent scheduling in the FJSP scenario and is closer to the real workshop production environment [12]. Cui et al. describe the long-term resource allocation problem as a stochastic game that maximizes expected returns, in which each drone becomes a learning agent and each resource allocation scheme corresponds to an action taken by the drone. Then, a multi-agent reinforcement learning (MARL) framework is developed, where each agent finds its optimal strategy based on its local observations of use learning [13]. In 2021, Farhadi et al. proposed a two-time-scale framework that combines optimization of service (data and code) placement and request scheduling within the storage, communication, computing, and budget constraints. By analyzing the difficulty of various situations to fully describe the complexity of the problem, a polynomial time algorithm is developed to achieve a constant factor approximation under certain conditions [8]. In the same year, Han et al. used the CMMAC algorithm to share a centralized state value function with multiple agents, and each actor only observed the local state [14]. However, the above paper does not consider the dynamic and time periodicity of the task queue in the server in the distributed scenario, so the timing relationship between inputs is not considered in the mentioned scheme.

To sum up, the existing computing resource allocation and scheduling scheme only analyzes the existing system status upon the arrival of each task and selects the node server to process the task through model judgment, without considering the time sequence relationship in the updating process of the task queue of the node server. The CMARAC algorithm proposed in this paper adopts the method of combining the recursive network and strategy to schedule each task and makes full use of the timing relationship of the task queue of the edge server node in each task scheduling process. Compared with the traditional scheme network, the convergence speed is faster and fewer data is required in the training process.

3 Scheduling Problem

3.1 Problem Definition

We consider the task scheduling process in the edge cluster environment and use the system throughput rate of the task scheduling process as one of the indicators to measure the algorithm effect, that is, the proportion of tasks completed on schedule in the total of all tasks during the long-term operation of the system. Given edge cluster $\mathcal{M} = \{1, 2, \ldots, M\}$, the index is m. To better complete the task scheduling process, each cluster is assigned a master responsible for managing cluster tasks. The nodes of each cluster are $\mathcal{N}_m = \{1, 2, \ldots, N_m\}$ is managed by master m. In each edge cluster, all nodes are represented as $\mathcal{N} = \{1, 2, \ldots, N\}$. All the masters are responsible for distributing the tasks allowed to be processed to the managed edge nodes or cloud. All nodes of an edge cluster communicate through the local area network. The edge cluster master m arranges a queue to record the ID of the task, $\mathcal{Q}_m$, which represents all the tasks to be completed and the order of task scheduling. A queue of tasks to be processed q_σ is arranged on all servers for which tasks can be assigned. σ is less than or equal to the total number of servers in a cluster or edge cluster. Tasks in the queue are prioritized according to time requirements.

To illustrate the number of tasks that each node can handle, this paper records a matrix $\mathcal{D} = \{d_1, d_2, \ldots, d_T\}$, including d_T record a server, can handle task types of one-hot vector, which $T = M + \sum_{i=1}^{m} N_i$, said the total number of all except the cloud cluster server. Compared with edge servers and edge nodes, the cloud has sufficient computing resources and task processing capacity and can complete all kinds of task processing by default. Therefore, matrix $\mathcal{D}$ does not contain information about the cluster.

The task scheduling time in this paper is strictly constrained. To correspond to the time unit $t$required by task scheduling in the data set, the resource scheduling system adjusts the time needed for a scheduling task $\tau = \alpha t (\alpha \in (0, 1))$. In each τ, the master m of each edge cluster will distribute a task to the edge node $\mathcal{N}_m$ with sufficient computing resources, or to the cluster processing, and each task will consume the computing resources and network bandwidth of the edge or cloud. However, because the edge master is typically close to the end device but not the cloud, sending a task to the cloud may result in additional communication overhead. Therefore, This paper's algorithm will lower the priority of the cloud processing task. When the system schedules tasks based on time, if any task is not processed in time, the system will check and discard it at each τ.

In this paper, the optimization of the target for throughput, in the process of system scheduling system to complete tasks $\Omega = \sum_s^{\infty} \sum_{\sigma}^{T+1} \omega_s(q_\sigma)$, $\omega_s(q_\sigma)$indicates the number of tasks completed by the server σ in time s, and $T+1$ indicates the number of all servers in the system. However, with the increase of time, Ω will continue to increase, which is not conducive to the subsequent calculation and optimization of the model. Throughput $\Phi \in [0, 1]$, the said system within the required time to complete the ratio of the total number of jobs and tasks, the total number of tasks $\Omega' = \sum_s^{\infty} \sum_m^{M} \omega'_s(\mathcal{Q}_m)$. Where $\omega'_s(\mathcal{Q}_m)$ represents the number of tasks received by edge cluster master m in time s. Therefore, the system throughput rate can be expressed as:

$$\Phi = \frac{\Omega}{\Omega'} = \frac{\sum_s^{\infty} \sum_{\sigma}^{T+1} \omega_s(q_\sigma)}{\sum_s^{\infty} \sum_m^{M} \omega'_s(\mathcal{Q}_m)} \tag{1}$$

4 Algorithm Design

4.1 CMARAC Algorithm Framework

The main body of the CMARAC algorithm contains two time scales, namely, the time scale of resource allocation and the time scale of recursive network input. The algorithm hierarchy diagram is as follows:

CMARAC algorithm is applied to hundreds of distributed edge clusters. Traditional learning algorithms, such as DQN [15] and DDPG [16], usually use a centralized learning agent, but it is difficult to realize for edge clusters, because distributed master will lead to the explosion of scheduling action space [17]. In order to ensure the real-time scheduling, CMARAC adopts a decentralized way to schedule requests at the place where the requests arrive.

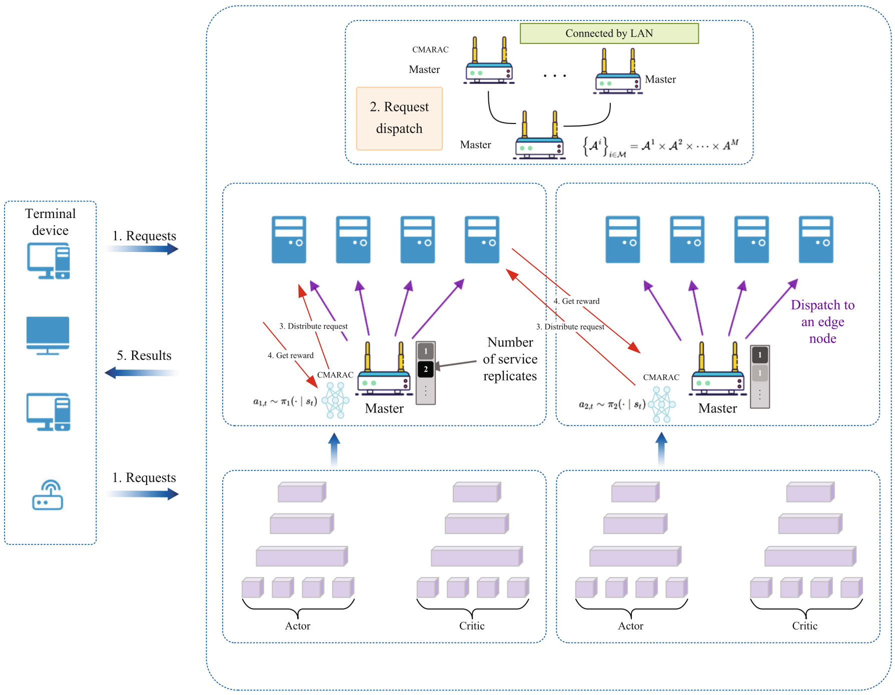

Connected by LAN
CMARAC
Master
2. Request dispatch
$\{\mathcal{A}^i\}_{i\in\mathcal{M}} = \mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^M$
Terminal device
1. Requests
5. Results
3. Distribute request
4. Get reward
$a_{1,t} \sim \pi_1(\cdot \mid s_t)$
$a_{2,t} \sim \pi_2(\cdot \mid s_t)$
Number of service replicates
Dispatch to an edge node
Actor
Critic

Fig. 1. Distributed request scheduling process based on CMARAC

We consider a typical distributed edge cluster scenario for request distribution. The complete request scheduling process is shown in Fig. 1. (i) the terminal device sends a request to the nearest edge cluster $\mathcal{M}$, and master m of the edge cluster receives the request and adds it to the request queue $\mathcal{Q}_m$. (ii) After completing the previous request scheduling process in queue $\mathcal{Q}_m$, the task is taken out of the queue. In addition, an agent of CMARAC deployed in the master is used to complete the scheduling process of the request, and finally the task is assigned to the appropriate node server for processing.

4.2 Identification of Temporal Characteristics of the Request

In a cluster environment, the arrival process of external requests usually follows a periodic rule. When Alibaba tracked the process of processing requests in their server cluster in 2020, it found the following rule: In a day, the situation of the server submitting tasks presents a periodic change with the change of time. Capturing such a change of task submission plays an important role in the scheduling process of external requests. The system can make preparations for task scheduling by predicting the arrival of peak request in advance.

Recurrent Neural networks (RNN) are neural networks with recurrent feedback connections that can be used to process data with sequential structure. The internal self-feedback of neurons gives them an inherent “depth” structure, which can obtain sufficient long-term dependence by using historical data [18,19]. Each time a new data is entered, the RNN passes the current input to the hidden layer for processing along with the state information of the previous time step, resulting in a new state output, which is passed to the next time step. This feedback connection enables RNNS to process sequence data with memory and the ability to consider previous information.

Long Short-Term Memory (LSTM) is a special recursive neural network (RNN), which solves problems such as gradient disappearance and gradient explosion in traditional RNN by introducing gating mechanism [20]. The cell structure of LSTM is shown below. In this paper, LSTM is used to extract the time characteristics of the request flow in the server cluster (Fig. 2).

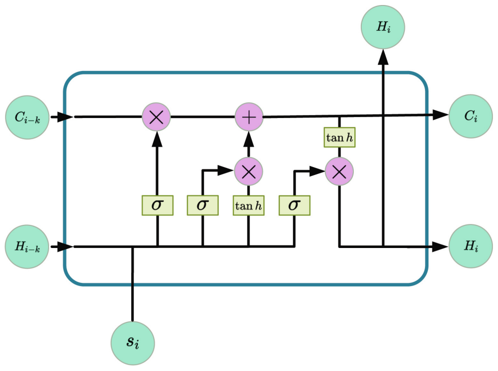


Fig. 2. Input cell diagram of system scheduling state

In this article, we overlay multiple LSTM layers onto the model to capture the time feature of higher level request flows. LSTM layer1 edge processing services in the cluster operation sequence $\mathbf{s} = [s_i, s_{i+k}, s_{i+2k}, ..., s_{i+ck}]$. Where s_i represents the system state at time i, k represents the time step, c represents the number of cells, and the system hidden state sequence $\mathbf{H} = [H_i, H_{i+k}, H_{i+2k}, ..., H_{i+ck}]$ at each time point is calculated. The hidden layer state sequence is then input to LSTM layer 2 to calculate the hidden layer state sequence $\mathbf{H}'$. Finally, the system state $\mathbf{H}_{i+ck}$ containing the past to present time characteristics is output.

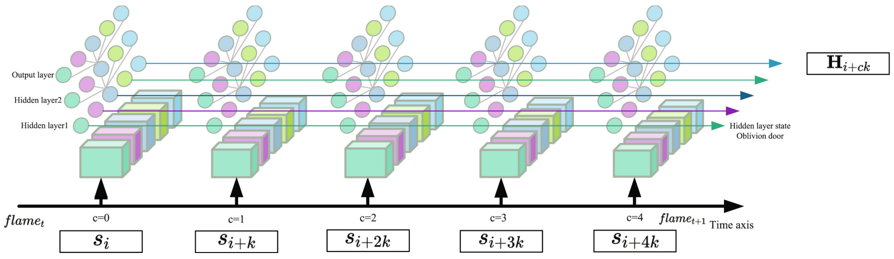


Fig. 3. Request timing feature recognition based on LSTM

Figure 3 shows the identification process of request features. The time axis of request arrival contains a time series of five time points, corresponding to $c = 0, 1, 2, 3, 4$, respectively. This time series corresponds to the system state characteristics corresponding to the first five time points including the current time. The final output $\mathbf{H}_{i+ck}$ is a combination of the system state characteristics of the whole period of time series, can effectively capture the timing characteristics of the system at five time points of the request, so as to make a better task scheduling scheme. Specific parameters are calculated as follows:

$$f_i = \sigma\left(W_{gf} s_i + b_{jf} + W_{hf} H_{(i-k)} + b_{hf}\right) \tag{2}$$

The calculation mode of f_i in Formula (2) represents the forgetting gate, W_{gf} and W_{hf} correspond to the weight parameters of the state input and hidden state input respectively, b_{jf} and b_{hf} represent the offset after the state input and hidden input respectively, s_i is the system state under the condition that the time corresponds to i. $H_{(i-k)}$ is the hidden state input, and σ is the sigma activation function.

$$I_i = \sigma\left(W_{gj} s_i + b_{jj} + W_{hj} H_{(i-k)} + b_{hj}\right) \tag{3}$$

$$g_i = \tanh\left(W_{jg} s_i + b_{jg} + W_{hg} H_{(i-k)} + b_{hg}\right) \tag{4}$$

$$C_i = f_i C_{(i-k)} + I_i g_i \tag{5}$$

The formulas for I_i, g_i, and C_i represent the update gate, W_{gj} and W_{hj} correspond to the weights of state inputs and hidden state inputs, b_{jj} and b_{hj} represent their respective offsets, and tanh represents the tangent function. Finally, the sequential state of the cell will be determined jointly by the calculated output of the forgetting gate and the output of the updaters in Formula (2), and the previous state $C_{(i-k)}$ will be transformed into the current state C_i.

$$o_i = \sigma\left(W_{go} s_i + b_{jo} + W_{ho} H_{(i-k)} + b_{ho}\right) \tag{6}$$

$$H_i = o_i \tanh\left(C_i\right) \tag{7}$$

Formula (6) represents the output gate, where W_{go} and W_{ho} are the weight parameters corresponding to the input state and hidden state of the system respectively, b_{jo} and b_{ho} are the offset respectively, and o_i is the calculation result of the output gate. The final output H_i from LSTM is determined by both the output gate and the cell update state C_i, as described in Formula (7). Finally, we will get the system state at the time step $i + ck$ and input it into the actor network and critic network.

Figure 4 shows the composition of CMARAC algorithm. On the whole, CMARAC algorithm is composed of two parts. The first part is responsible for collecting the request state information of the system in a period of time series and the time sequence relationship between the states, and finally obtaining the system state $\mathbf{H}_i$ containing the time sequence features. $\mathbf{H}_i$ is input into the Actor and Critic network, the state value is calculated, and the appropriate node in the edge cluster is selected to process the next task in the request queue $\mathcal{Q}_m$.

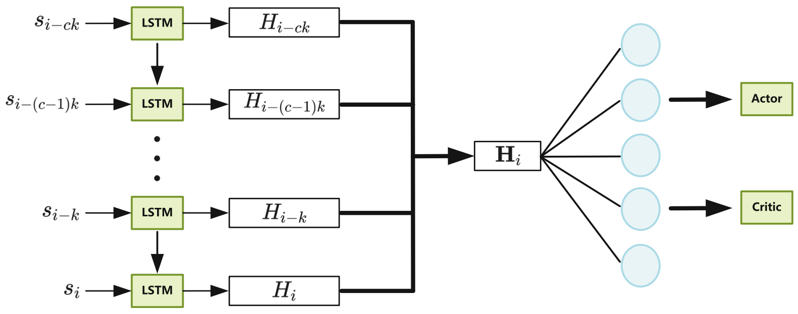


Fig. 4. CMARAC model structure diagram

4.3 Task Scheduling Procedure

In recent years, a revolutionary breakthrough has been made in the field of multi-agent deep reinforcement learning, which has been successfully applied to a variety of complex scenarios, proving that pure cooperative tasks can be represented by monotone hybrid networks, and agents based on neural networks can learn their actions in multi-agent environments through interaction [21].

In contrast to the general RL training process, DQN relies on a small batch of random sampling to experience the transition in the replay buffer, rather than just selecting a single transition. In addition, the target network with the same network structure as $Q(s_t, a_t)$ is used to reduce the correlation. And the target network is only updated at certain intervals.

DQN cannot be directly applied to solve continuous value control problems, so we adopt the most popular actor-critic method to deal with the computing resource scheduling problem in the edge cluster scenario. Specifically, actor-critic consists of a DNN acting as an actor network and a DQN called critic network [22]. The Actor represents our mapping function $\pi\left(s_t \mid \theta^\pi\right)$, and the critic performs the function $Q\left(s_t, a_t \mid \theta^Q\right)$. Actor can generate the optimal action a_t based on state s_t.

This chapter will introduce the multi-agent reinforcement learning method based on recursive network. Deploy critic network in the cloud, and deploy an actor network in each master at the edge. Usually, an agent does not fully understand all the knowledge of the environment state $\mathcal{S}$, so we express the process of each master to schedule the tasks in the queue as a Markov game Γ for cloud edge cluster, $\Gamma = \left(\mathcal{M}, \mathcal{S}, \left\{\mathcal{A}^i\right\}_{i\in\mathcal{M}}, \mathcal{P}, \left\{R^i\right\}_{i\in\mathcal{M}}, \gamma\right)$. Where $\mathcal{M} = \{1, 2, \ldots, M\}$ represents the agent of M edge clusters, $\mathcal{S}$ represents the state space, $\mathcal{A}^i$ represents the action taken by the ith master in the edge cluster, that is, the first task in the queue is selected and dispatched to a node, $\mathcal{P}$ represents the selection probability of each action, R^i represents the reward obtained after the ith master takes the action. We adopt the mechanism that all agents share the same reward function to deal with the computational resource scheduling problem proposed in this paper. $\gamma \in (0, 1)$ is a discount factor that can be used to control the influence of an agent on other agents. When the agent considers future rewards, it can take into account the actions and strategies that other agents may take.

In this case, the discount factor can be used to measure the agent's impact on future rewards, and thus influence its behavior and strategy.

$\mathcal{S}$ represents the state space, where in each time unit t, a state value $s_{t,m}$ is constructed for node σ in each edge cluster m. The composition of the state value is shown in the following table:

Table 1. Environment status

State	Description
$q_{\sigma,t}$	The queue of tasks to be processed in Node σ
$\mathcal{Q}_{m,t}$	The queue of tasks to be completed in Master m
L	Transmission delay between the cloud and the edge cluster master
Re	Remaining CPU and storage space resources of each cluster node N_m
St	N_m service type deployed on each cluster node
N	Number of nodes N_m in an edge cluster m

Table 1 shows the status information of time unit t. $q_{\sigma,t}$ represents the task queue that has not been processed since time t in node σ; $\mathcal{Q}_{m,t}$ represents the task queue information that has not been dispatched since time t in master m; L represents the transmission delay between the cloud and the edge cluster master. Re represents the remaining CPU and storage space resources of each cluster node N_m at time t, St represents the service type deployed by each cluster node N_m, and N represents the number of node N_m in an edge cluster m. The above parameters are combined into $s_{t,m}$. In order to concentrate the status information and facilitate the update of critic network, We maintain a global state $s_t \in \mathcal{S}$, which not only contains the state information of all cluster nodes N_m, but also includes the task queue information $\mathcal{Q}_{C,t}$ in the cloud.

The action space of Markov game $\{\mathcal{A}^i\}_{i\in\mathcal{M}}$ is expressed as $\{\mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^M\}$, represents the action space of the state space of the agent deployed in each edge cluster m. For edge cluster, we regard all available edge nodes as a resource pool, that is, the mutual cooperation among the masters in the enabled state. All $\mathcal{A}^i$ contains the number of all available edge nodes and the cloud $\{1, 2, 3, \ldots, N_1 + N_2 + \cdots + N_m + M\}$, and the sum of all edge nodes, masters, and the cloud is the size of the action space $\mathcal{A}^i$. In particular, when $\mathcal{A}^i = 0$, the cloud handles the current request. We only allow each master agent to schedule one request within time unit t.

Generally speaking, the master of an edge server cluster needs to balance the system throughput rate and resource scheduling costs when allocating resources to server nodes. Therefore, a reward function is needed to teach the agent to learn the potential resource allocation plan. The return function derived from ensuring the throughput rate of the system and realizing the load balancing of the cluster can be obtained:

$$r_1 = e^{\mu}, \mu \in [0, 1] \tag{8}$$

$$r_2 = e^{-\beta}, \beta \in [\frac{1}{2}, 1] \tag{9}$$

$$R = r_1 r_2 \tag{10}$$

Reward function contains two parts, the first part of the throughput rate of the system, as shown in Formula (8). The value range of r_1 is $[e^{-1}, 1]$, μ represents the system throughput rate at time $[\lambda t, \lambda(t+1)], \lambda = 1, 2, \ldots, n$, λ represents the time step, that is, the system throughput rate within a period of time is calculated after a period of time unit t. The second part considers cluster load balancing, as shown in Formula (9), $\beta = 1/(1 + e^{-\xi})$ is the variance of normalized node resource consumption ξ. Finally, the total reward R is the product of r_1 and r_2.

In an edge cluster environment, the state transition probability of multi-agent reinforcement learning is the joint state transition probability of all agents, which is expressed as follows:

$$P\left(s_{t+1} \mid s_t, s_{t-1}, \cdots, s_1, s_0\right) = P\left(s_{t+1} \mid s_t\right) = p\left(s_{t+1} \mid s_t, a_{1,t}, a_{2,t}, \ldots, a_{n,t}\right) \tag{11}$$

Formula (11) shows that the state s_t conforms to the Markov property, that is, the state of the next time node only depends on the current state and an action made in the current state, and the state is the environment that the system can achieve through some way [23]. In multi-agent reinforcement learning, we construct an observable Markov model [24]. The status s_{t+1} of the next point in time is only related to the status s_t of the current point in time and the action $\{a_{1,t}, a_{2,t}, \ldots, a_{n,t}\}$ taken by all agents.

The state value function of agent in edge cluster master is the function of joint strategy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, which can be defined as:

$$\pi(a_t|s_t) = \prod_{i \in \mathcal{M}} \pi_i\left(a_{i,t}|s_{i,t}\right) \tag{12}$$

Therefore, for every joint strategy π and state $s_t \in \mathcal{S}_t$, the common state value function $V_{m,\pi}$ of agents can be derived by minimizing Bellman equation:

$$V_{m,\pi}(s_t) = \mathbb{E}\left[\sum_{t \geq 0} \gamma R\left(s_t, a_{m,t}, s_{t+1}\right) \mid a_{m,t} \sim \pi_m\left(\cdot \mid s_t\right), s_0 = s\right] \tag{13}$$

The multi-agent collaborative resource scheduling algorithm proposed by us is designed based on the actor-critic algorithm framework, including the actor network deployed in different edge clusters and the critic network deployed in the cloud. The coordinated learning method of centralized critic and distributed actor is adopted. When training critic network, all actors share a centralized state value function, while in the training and reasoning process of distributed actor network, each actor only observes the local state.

The Critic network is divided into target value network θ' and value network θ. We used the target value network θ' to calculate TD (Temporal Difference) errors, and kept updating them in the training process, but at a slow speed.

Generally, parameters of Critic network θ will not be copied to target value network θ' until a certain time interval is required, so the objective function V of target network can be expressed as Formula (14):

$$V\left(s_{t+1};\theta',\pi\right)=\sum_{m\in\mathcal{M},t\geq 0}\pi\left(a_{m,t}\mid s_{m,t}\right)\left(R+\gamma V'\left(s_{m,t+1}\right)\right)\tag{14}$$

$$\mathcal{L}\left(\theta'\right)=\left(R+\gamma V'\left(s_{m,t};\theta'\right)-V\left(s_{m,t};\theta'\right)\right)^2\tag{15}$$

The objective function in Formula (14) is the weighted sum of agent action value and reward function under different strategies. The weight is the probability of edge cluster m taking relative action $a_{m,t}$ against state $s_{m,t}$ at time point t. For all edge clusters $\mathcal{M}$ in edge server cluster m, each time t has a unique state value $\{V\left(s_{m,t}\right),m\in\mathcal{M}\}$. Formula (15) represents the loss function of the target network, $V\left(s_{m,t};\theta'\right)$ represents the value estimate of the target value network for the current state, $V'\left(s_{m,t};\theta'\right)$ represents the value estimate of the target value network for the next state.

We define an advantage function $A\left(s_{t,m},a_{m,t}\right)$ and calculate gradient ∇_a to update actor networks, where the advantage function $A\left(s_{t,m},a_{m,t}\right)$ is shown in Formula (16):

$$A\left(s_{t,m},a_{m,t}\right)=R+\gamma V'\left(s_{m,t};\theta'\right)-V\left(s_{m,t};\theta'\right)\tag{16}$$

γ is the discount factor, $V\left(s_{m,t};\theta'\right)$ represents the value estimate of the target value network for the current state, $V'\left(s_{m,t};\theta'\right)$ represents the value estimate of the target value network for the next state, R represents the reward after taking action $a_{m,t}$ in the current state, and then calculate the gradient used to update the actor network, as shown in Formula (17):

$$\nabla_a=\nabla\log\pi\left(a_{m,t}\mid s_{t,m}\right)A\left(s_{t,m},a_{m,t}\right)\tag{17}$$

where $\pi\left(a_{m,t}\mid s_{t,m}\right)$ is the strategy of corresponding state $s_{t,m}$ in edge cluster m which takes related action $a_{m,t}$, after calculating the gradient of actor network, the network can be updated. The overall algorithm flow is as follows:

The above algorithm flow is the whole process of training and scheduling of CMARAC. In each system scheduling time unit, each edge cluster master will realize the scheduling of the first task in queue $\mathcal{Q}_m$, and eject it from the queue and distribute it to a node in the cluster for processing according to the action $a_{m,t}$ output by the actor network. Each time to complete the scheduling process of a specific time, according to the recorded status, action and training parameters, calculate the loss function value $\mathcal{L}\left(\theta'\right)$and gradient ∇_a to update the actor network and critic network.

5 Experiment and Analysis

In this paper, the optimization of the target for the throughput, in the process of system scheduling system to complete tasks $\Omega=\sum_s^{\infty}\sum_{\sigma}^{T+1}\omega_s(q_\sigma)$, including $\omega_s(q_\sigma)$ said the server σ number of tasks in a timely manner.

Algorithm 1. CMARAC training and scheduling process

```
1: Initialize the system environment and neural network structure
2: for $\tau$ in 1, 2, 3, ... do
3:     for each $m$ in $\mathcal{M}$ do
4:         Update task queue $\mathcal{Q}_m$ in edge cluster $m$
5:         Update CPU and storage resource $Re$ of each node in cluster $m$
6:         Remove a task from the task queue $\mathcal{Q}_m$
7:         Action $a_{m,t}$ is taken to assign tasks by formula 12
8:         if $\tau \% reward\ cycle$ then
9:             Formulas (8), (9), and (10) were used to calculate the current cluster return $R_m$
10:            Formula (8): $r_1 = e^{\mu}, \mu \in [0, 1]$
11:            Formula (9): $r_2 = e^{-\beta}, \beta \in [\frac{1}{2}, 1]$
12:            Formula (10): $R = r_1 r_2$
13:            Calculate the dominant functions $A(s_{t,m}, a_{m,t})$ and gradient $\nabla_a$ using formulas (16) and (17)
14:            Formula (16): $A(s_{t,m}, a_{m,t}) = R + \gamma V'(s_{m,t}; \theta') - V(s_{m,t}; \theta')$
15:            Formula (17): $\nabla_a = \nabla \log \pi(a_{m,t} \mid s_{t,m}) A(s_{t,m}, a_{m,t})$
16:            Record the critic network training parameters $[s_t, V(s_{t+1}; \theta', \pi)]$
17:            Record the actor network training parameters $[s_{t,m}, a_{m,t}, A(s_{t,m}, a_{m,t}), R_m]$
18:        end if
19:    end for
20:    The loss function $\mathcal{L}(\theta')$ and gradient $\nabla_a$ are calculated using formulas (14) and (15)
21:    Update actor and critic network parameters
22:    Formula (14): $V(s_{t+1}; \theta', \pi) = \sum_{m \in \mathcal{M}, t \geq 0} \pi(a_{m,t} \mid s_{m,t})(R + \gamma V'(s_{m,t+1}))$
23:    Formula (15): $\mathcal{L}(\theta') = (R + \gamma V'(s_{m,t}; \theta') - V(s_{m,t}; \theta'))^2$
24: end for
```

5.1 Experimental Setup and Data Set

The data set of this experiment is Alibaba cluster tracking data set [25], which makes detailed statistics on the workload tracking data collected in the production MLaaS cluster of more than 6000 GPUs for two months. Alibaba Cluster Trace data set comes from Alibaba Cluster Trace Program, which makes a comprehensive analysis of Alibaba's large-scale workload tracking and reveals the benefits of GPU sharing in the production of GPU data centers [26]. It is widely used in academia and industry. We modified these server trace data sets to external request data sets, and the data structure of this data set and its examples are shown in Table 2:

We used task_type, start_time, end_time, plan_cpu and plan_mem in the data set as experimental data. Other data were not used in this experiment. This is because this experiment deals with the scheduling of different types of external requests, one for each service, and the service deployment status of the server is reflected in deploy_state.

Table 2. Data items and examples in the data set

Columns	Example Entry
task_name	task_ODk2MzU0ODg1MTY5MTExNTUwMg==
inst_id	9
job_name	j_18443
task_type	3
status	Terminated
start_time	86442
end_time	86446
plan_cpu	300
plan_mem	0.39

5.2 Experimental Results and Analysis

In order to evaluate the effectiveness of CMARAC algorithm and the advantages compared with other algorithms, this paper obtained some high quality performance of CMARAC algorithm from different dimensions through 5 comparison experiments with other algorithms, and carried out optimization experiments on the parameters of CMARAC algorithm.

5.2.1 Overall Performance

Table 3. Scheduling effect comparison between CMARAC and random scheduling, greedy policy and cMMAC

Contrast round	Random scheduling	Greedy strategy	cMMAC	CMARAC
1	0.277	0.679	0.808	0.819
2	0.316	0.674	0.81	0.842
3	0.317	0.673	0.823	0.85
4	0.33	0.734	0.824	0.849
5	0.337	0.688	0.83	0.85
6	0.336	0.694	0.832	0.854
7	0.314	0.651	0.835	0.854
8	0.347	0.695	0.833	0.853
9	0.434	0.661	0.839	0.854
10	0.315	0.666	0.84	0.852
11	0.304	0.662	0.84	0.851
12	0.348	0.698	0.842	0.854
13	0.342	0.653	0.843	0.856
14	0.36	0.7	0.842	0.854
15	0.331	0.712	0.844	0.855

This experiment adopted cluster-trace-Gpu-V2020, the Alibaba cluster tracking data set published by Alibaba Group. We compared the scheduling effects of random scheduling, greedy strategy, cMMAC [12] and CMARAC algorithm. There were 15 rounds of experiments. The experimental data used in each round contains all the requests in the 50,000 time units, which are different for different rounds. The same cloud edge system adopts different scheduling algorithms respectively, and the throughput rate after each algorithm is compared. The comparison effect is shown in Table 3. The best result of each instance is shown in boldface.

It can be seen from the above comparison that the scheduling effect of the proposed CMARAC is generally better than that of random scheduling, greedy strategy and cMMAC. Taking the throughput rate of all requests in the same time period with a range of 50,000 time units as the index, the CMARAC algorithm improves 1.545 times compared with random scheduling, 24.5% compared with greedy policy scheduling, and 2.1% compared with baseline cMMAC algorithm scheduling. This indicates that CMARAC has a higher scheduling effect than other algorithms when facing a large number of request data with timing characteristics.

Request timing feature identification is a key step in CMARAC scheduling process, which determines whether the system can learn the timing feature in the process of request arrival, and thus determines the scheduling performance of CMARAC. The following experiments show that the convergence rate of CMARAC algorithm is faster than that of baseline cMMAC algorithm.

Figure 5 shows the training process of CMARAC algorithm and cMMAC algorithm. The training data adopts all requests within 50000 time units in cluster-trace-gpu-v2020 data set, and a total of 50 rounds of training are conducted. Each round of training includes 50 calculations of loss value and network updates. The label data in the figure is the median of loss value in each round

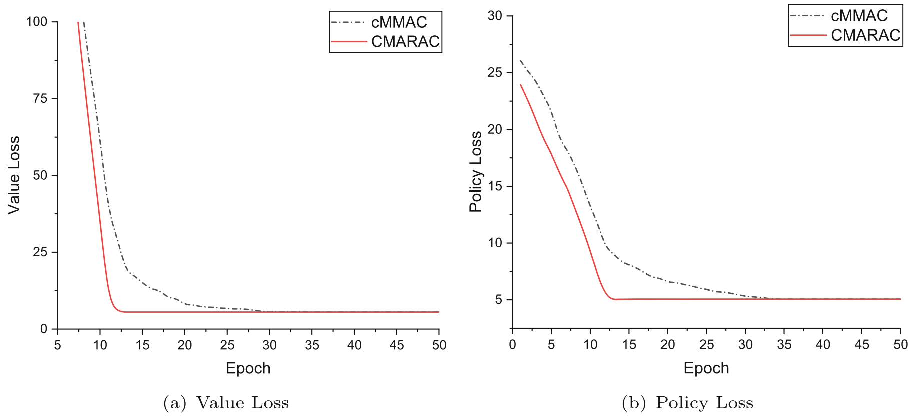


Fig. 5. Comparison of network convergence effect of CMMARAC and cMMAC algorithm, (a) represents the convergence of loss value of actor network, and (b) represents the convergence of loss value of critic network. In the figure, the algorithm achieves convergence of loss value with as few training cycles as possible, which indicates that the algorithm has better performance.

of data. The hidden layer nodes of both algorithm cMMAC and CMARAC network are [128, 32], and the number of cells participating in the comparison of CMARAC algorithm is 4.

In the CMARAC algorithm, the request feature recognition network proposed by us adopts the LSTM recursive network, which can capture the time sequence dependence in the input sequence, remember the past information, and apply it to the current context. By iterating through multiple time steps, LSTM can gradually build an abstract representation of the sequence data, capturing key patterns and features in the input. In cloud edge systems with significant periodic request arrival processes, using LSTM to extract temporal features from requests can help understand and model the temporal patterns and behaviors of requests. By utilizing LSTM's sequential modeling capabilities, we can better understand and process timing data related to computing resource scheduling.

Figure 5 respectively shows the change of loss value of CMARAC and cMMAC with the training rounds. For the calculation process of the loss value of actor network, the loss value of CMARAC is always lower than that of cMMAC. When the training cycle is 1, the loss value of CMARAC is 304 and the loss value of cMMAC is 302. When the training cycle is 13, CMARAC reaches the state of convergence. cMMAC converges, and the final loss converges to about 5.53. For critic network, the initial loss value of CMARAC was 23.97, and reached convergence in the 13th round of training; the initial loss value of cMMAC was 26.08, and reached convergence in the 34th round, with the final loss converging to about 5.07.

Compared with cMMAC algorithm, the convergence speed of CMARAC is 1.69 times faster. This is because LSTM can capture the timing characteristics in a large number of requests, so as to achieve the best scheduling effect in fewer training rounds. Moreover, the calculation method of our return function considers the scheduling failure rate and load balancing rate in a period of time. The network has a better scheduling effect. Compare the throughput of CMARAC and cMMAC in the training process, as shown in Fig. 6.

It can be seen that CMARAC showed a high system throughput rate of 0.852 in the fourth round, and reached a convergence state of 0.855 in the eighth round. The system throughput rate based on cMMAC algorithm converges to 0.854 in the 13th round. Taking the system throughput rate as the indicator, the convergence rate of CMARAC is 62.5% faster than that of cMMAC, and the system throughput rate of Cmarac is 0.1% higher than that of cMMAC when it reaches the convergence state.

5.2.2 Load Balancing Effect

In edge clusters, task load balancing is an important aspect of computing resource allocation [27]. Load balancing distributes tasks and traffic evenly to edge nodes to ensure load balance among each node and avoid overloads on individual nodes, thus improving overall performance and scalability.

As scheduling tasks and recorded the resource usage of each edge node. In the experiment, it is assumed that each task occupies corresponding resources

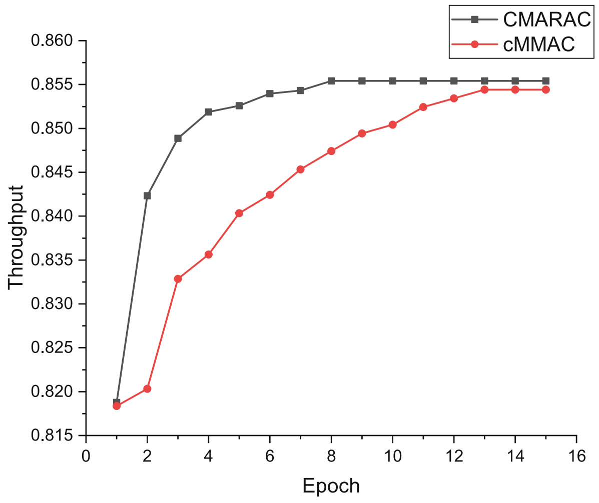
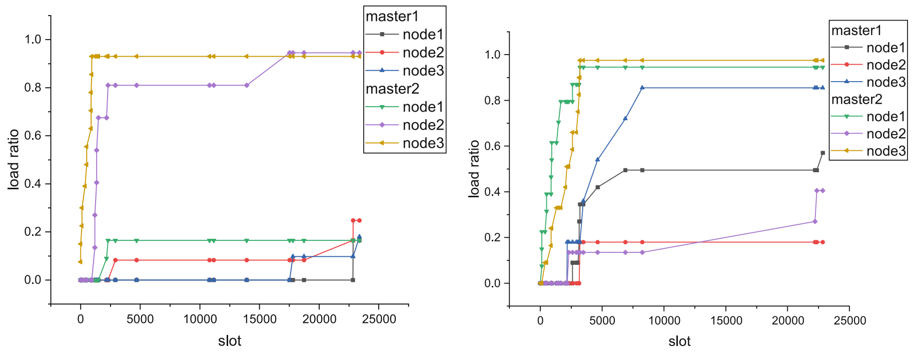


Fig. 6. Comparison of throughput between CMARAC and cMMAC algorithms in the training process

after being distributed to a node, and the load rate of the node will keep a rising trend. The number of nodes of the network hidden layer of cMMAC is [128, 32], the number of nodes of the network hidden layer of CMARAC is [64, 16], and the number of cells is 7. The operating environment is two edge clusters. Each edge cluster contains three edge nodes, and the service deployment of each node is fixed. The changes of resource load rates of six edge nodes were recorded in the test, as shown in Fig. 7:



(a) cMMAC Load Ratio (b) CMARAC Load Ratio

Fig. 7. Changes in the algorithm load rate of each node in the edge cluster. (a) represents the change of node load rate based on cMMAC algorithm, and (b) represents the change of node load rate based on CMARAC algorithm.

There are two reasons for the load difference between nodes. (1) Different nodes deploy different kinds of services. Since a service can only be used to process one kind of request, the quantity difference of different kinds of request data will cause the node load difference. (2) When a service is deployed on different nodes at the same time, the algorithm considers the resource usage of the node, the ability of the node to process the request, the communication cost between the node and the master and other factors, and then selects one of the nodes to process the current request and allocate resources, which will also cause the difference in node load. In Fig. 7(a), the load rate of the three nodes of the cluster where master1 resides and node1 of the cluster where master2 resides are all at a low level, while the overall load rate of node2 and node3 of the cluster where master2 resides is high, and the load rate reaches 0.8 or above in a short time. In Fig. 7(b), node3 of the cluster where master1 resides and node1 and node3 of the cluster where master2 resides have high load rates, while the load rates of other nodes are low.

By calculating the load balancing index of the system using different algorithms for resource scheduling, namely the standard deviation of node resource consumption, we can compare the load balancing effect of different algorithms. The comparison effect is shown in Fig. 8:

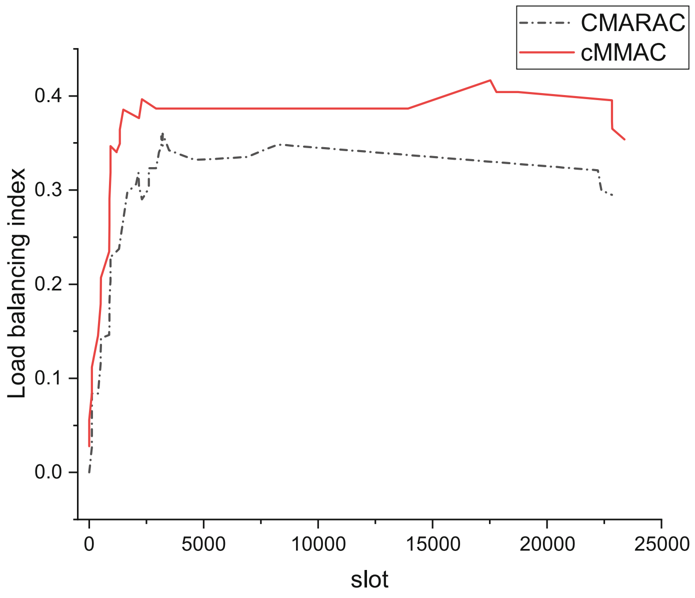


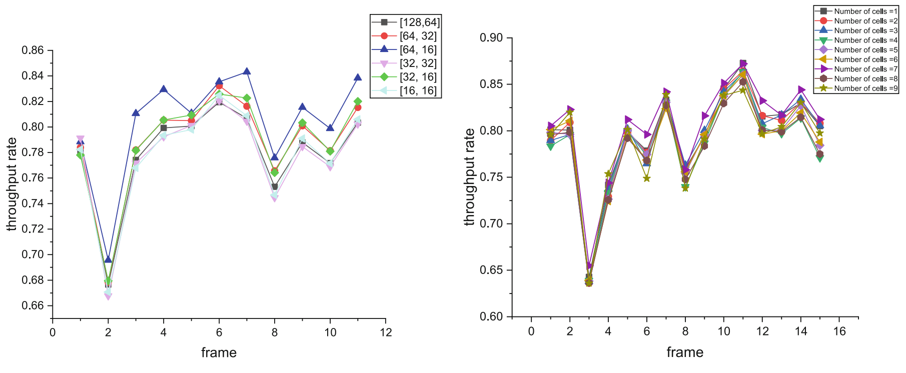
Fig. 8. Comparison of load balancing index

It can be seen that the standard deviation of the system resource consumption using CMARAC algorithm is smaller than that using cMMAC algorithm in most of the scheduling time, with the maximum standard deviation being 0.360 and

0.417. In the computing resource scheduling scenario of distributed edge clusters, the load status may dynamically change with the increase or decrease of external request traffic. LSTM can adjust the model adaptively according to the real-time observed load information. By learning the timing pattern of the load, the LSTM network can adjust the decision-making strategy in time to adapt to the requirements of different load states. Therefore, compared with cMMAC without LSTM network model, CMARAC achieves better cluster load balancing effect.

5.2.3 Parameter Optimization Experiment

Parameter optimization is a very important problem in CMARAC, because it determines the performance and training efficiency of the algorithm. Optimizing parameters can make the model better adapt to the specific environment and improve its generalization ability for various scenarios. For example, the number of hidden layers in the recursive network, the number of neurons in each hidden layer, the network structure of actor-critic network, payback function and so on. Based on the throughput rate of task scheduling process, this paper conducts relevant experiments on the influence of the number of neurons in the hidden layer of recursive network on scheduling effect in the process of time sequence feature recognition. 25000 time units are taken as the interval, 1 frame is defined as 25000 time unit, and the request in Alibaba tracking data set is taken as the scheduling target. The number of cells is 4 and the number of hidden layers is 2. Throughput rate is used to measure the effectiveness of the algorithm, as shown in Fig. 9:



(a) Influence of neurons in the hidden layer (b) Effect of cell on scheduling effect

Fig. 9. Changes in the algorithm load rate of each node in the edge cluster. (a) represents the change of node load rate based on cMMAC algorithm, and (b) represents the change of node load rate based on CMARAC algorithm.

According to Fig. 9, when other parameters of the recursive network and the number of hidden layers remain unchanged, the number of neurons in different networks corresponds to different throughput rates. On the whole, when the neuronal structure is [64,16], the system throughput rate is the highest in different time periods. In the whole experiment, the average system throughput rate is 0.80. When the neuronal structure is [32, 32], the average system throughput is the lowest 0.77, while when the neuronal structure is other structures in the figure, the average system throughput is concentrated around 0.78. It can be clearly seen from the figure that when the hidden layer is [64, 16], the line graph is at the top and the system throughput is the highest. Therefore, the optimal neuron structure of the hidden layer is [64, 16].

We continue to optimize the structure of the LSTM part of the neural network, and explore the influence of the number of cells on the system request throughput rate when 1 frame is 25000 scheduling time unit and the number of neurons in the hidden layer is [128, 32].

We can see that when the number of cells is 7, the best effect is achieved in the test area. After taking Alibaba tracking requests of different time periods and inputting them into the system, the average system throughput rate can be obtained at 0.80. When the number of cells is other numbers, the average system throughput rate is between [0.78, 0.79]. Therefore, the experiment proves that the task scheduling effect of the system is the best when the number of cells is 7. When the number of cells is too small, the network cannot capture the time sequence feature in the process of request arrival; when the number of cells is too large, the network appears the phenomenon of overfitting, which leads to the degradation of the scheduling performance of the model.

6 Summary

In order to solve the problem of periodic computing resource scheduling for external requests, this paper proposes the CMARAC algorithm. Its main idea is to take the system state including multiple time nodes in the present and the past and the request information in the current server cluster as the input, and use LSTM to extract the time characteristics of the request flow in the server cluster. Finally, it is output to the actor-critic network, and the most suitable edge node is selected for processing, so as to achieve the function of resource scheduling for external requests. Experiments show that the proposed multi-agent cooperative computing resource scheduling algorithm (CMARAC) for periodic task scenarios has better scheduling effect than random scheduling algorithm, greedy strategy and cMMAC. In addition, compared with the baseline, the request timing feature recognition function proposed in this paper can capture the change of task submission, and the convergence speed is faster. Compared with the actor-critic algorithm alone, the system computation cost is less, and the scheduling scheme produced by greedy strategy judgment is better. CMARAC can schedule computing resources when multiple edge clusters are combined, improving the processing efficiency of external requests. Subsequently, we plan to further improve this algorithm and perfect the time step between the input time series of the algorithm.

References

1. Wang, R., Lai, J., Zhang, Z., Li, X., Vijayakumar, P., Karuppiah, M.: Privacy-preserving federated learning for internet of medical things under edge computing. IEEE J. Biomed. Health Inform. **27**, 854–865 (2022)
2. Deng, S., Zhao, H., Fang, W., Yin, J., Dustdar, S., Zomaya, A.Y.: Edge intelligence: the confluence of edge computing and artificial intelligence. IEEE Internet Things J. **7**(8), 7457–7469 (2020)
3. Khan, W.Z., Ahmed, E., Hakak, S., Yaqoob, I., Ahmed, A.: Edge computing: a survey. Futur. Gener. Comput. Syst. **97**, 219–235 (2019)
4. Zhang, J., Chen, B., Zhao, Y., Cheng, X., Hu, F.: Data security and privacy-preserving in edge computing paradigm: survey and open issues. IEEE Access **6**, 18209–18237 (2018)
5. Lu, C., Ye, K., Xu, G., Xu, C.Z., Bai, T.: Imbalance in the cloud: an analysis on alibaba cluster trace. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 2884–2892. IEEE (2017)
6. Tianqing, Z., Zhou, W., Ye, D., Cheng, Z., Li, J.: Resource allocation in IoT edge computing via concurrent federated reinforcement learning. IEEE Internet Things J. **9**(2), 1414–1426 (2021)
7. Houssein, E.H., Gad, A.G., Wazery, Y.M., Suganthan, P.N.: Task scheduling in cloud computing based on meta-heuristics: review, taxonomy, open challenges, and future trends. Swarm Evol. Comput. **62**, 100841 (2021)
8. Farhadi, V., et al.: Service placement and request scheduling for data-intensive applications in edge clouds. IEEE/ACM Trans. Netw. **29**(2), 779–792 (2021)
9. Liu, B., Liu, C., Peng, M.: Resource allocation for energy-efficient MEC in NOMA-enabled massive IoT networks. IEEE J. Sel. Areas Commun. **39**(4), 1015–1027 (2020)
10. Chen, X., Zhu, F., Chen, Z., Min, G., Zheng, X., Rong, C.: Resource allocation for cloud-based software services using prediction-enabled feedback control with reinforcement learning. IEEE Trans. Cloud Comput. **10**(2), 1117–1129 (2020)
11. Wang, R., et al.: Multivariable time series forecasting using model fusion. Inf. Sci. **585**, 262–274 (2022)
12. Chen, Y., Liu, Z., Zhang, Y., Wu, Y., Chen, X., Zhao, L.: Deep reinforcement learning-based dynamic resource management for mobile edge computing in industrial internet of things. IEEE Trans. Industr. Inf. **17**(7), 4925–4934 (2020)
13. Cui, J., Liu, Y., Nallanathan, A.: Multi-agent reinforcement learning-based resource allocation for UAV networks. IEEE Trans. Wirel. Commun. **19**(2), 729–743 (2019)
14. Han, Y., Shen, S., Wang, X., Wang, S., Leung, V.C.: Tailored learning-based scheduling for Kubernetes-oriented edge-cloud system. In: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, pp. 1–10. IEEE (2021)
15. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
16. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
17. Wang, F., Wang, F., Liu, J., Shea, R., Sun, L.: Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, pp. 2499–2508. IEEE (2020)
18. Fei, J., Liu, L.: Real-time nonlinear model predictive control of active power filter using self-feedback recurrent fuzzy neural network estimator. IEEE Trans. Ind. Electron. **69**(8), 8366–8376 (2021)

19. Funahashi, K., Nakamura, Y.: Approximation of dynamical systems by continuous time recurrent neural networks. Neural Netw. **6**(6), 801–806 (1993)
20. Zheng, H., Lin, F., Feng, X., Chen, Y.: A hybrid deep learning model with attention-based conv-LSTM networks for short-term traffic flow prediction. IEEE Trans. Intell. Transp. Syst. **22**(11), 6910–6920 (2020)
21. Hu, J., Jiang, S., Harding, S.A., Wu, H., Liao, S.W.: Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. arXiv preprint arXiv:2102.03479 (2021)
22. Wang, L., Wang, K., Pan, C., Xu, W., Aslam, N., Hanzo, L.: Multi-agent deep reinforcement learning-based trajectory planning for multi-UAV assisted mobile edge computing. IEEE Trans. Cogn. Commun. Netw. **7**(1), 73–84 (2020)
23. Dorronsoro, B., Bouvry, P.: Improving classical and decentralized differential evolution with new mutation operator and population topologies. IEEE Trans. Evol. Comput. **15**(1), 67–98 (2011)
24. Littman, M.L: Markov games as a framework for multi-agent reinforcement learning. In: Machine Learning Proceedings 1994, pp. 157–163. Elsevier (1994)
25. Weng, Q., et al.: MLaaS in the wild: workload analysis and scheduling in large-scale heterogeneous GPU clusters. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), pp. 945–960. USENIX Association (2022)
26. Gao, W., et al.: Deep learning workload scheduling in GPU datacenters: taxonomy, challenges and vision. arXiv preprint arXiv:2205.11913 (2022)
27. Jena, U., Das, P., Kabat, M.: Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. J. King Saud Univ.-Comput. Inf. Sci. **34**(6), 2332–2342 (2022)