Intelligent Automation & Soft Computing

Tech Science Press

DOI: 10.32604/iasc.2023.038332
Article

Reliable Scheduling Method for Sensitive Power Business Based on Deep Reinforcement Learning

Shen Guo*, Jiaying Lin, Shuaitao Bai, Jichuan Zhang and Peng Wang

China Electric Power Research Institute, Beijing, 100192, China
*Corresponding Author: Shen Guo. Email: 1119312172@qq.com
Received: 08 December 2022; Accepted: 28 February 2023

Abstract: The main function of the power communication business is to monitor, control and manage the power communication network to ensure normal and stable operation of the power communication network. Communication services related to dispatching data networks and the transmission of fault information or feeder automation have high requirements for delay. If processing time is prolonged, a power business cascade reaction may be triggered. In order to solve the above problems, this paper establishes an edge object-linked agent business deployment model for power communication network to unify the management of data collection, resource allocation and task scheduling within the system, realizes the virtualization of object-linked agent computing resources through Docker container technology, designs the target model of network latency and energy consumption, and introduces A3C algorithm in deep reinforcement learning, improves it according to scene characteristics, and sets corresponding optimization strategies. Minimize network delay and energy consumption; At the same time, to ensure that sensitive power business is handled in time, this paper designs the business dispatch model and task migration model, and solves the problem of server failure. Finally, the corresponding simulation program is designed to verify the feasibility and validity of this method, and to compare it with other existing mechanisms.

Keywords: Power communication network; dispatching data networks; resource allocation; A3C algorithm; deep reinforcement learning

1 Introduction

Power network faces millions of households with high service requirements and strong social influence [1–5]. It is the solid backing of the rapid development of the country. The reliability of power communication network, which is the bearing net-work of power business, is closely related to the stable operation of power network. To ensure the continuity of power business and avoid large power outage accidents [6–11].

The main function of the power communication business is to monitor, control and manage the power communication network to ensure the regular and stable operation of the power communication network [12]. Current power communication business mainly includes power dispatch telephone control, communication equipment fault diagnosis and alarm, network equipment management, VLAN configuration, network performance and status chart analysis, port status monitoring and analysis, network topology management and many other functions [13–16]. The specific characteristics of the power communication business are shown in Table 1.

Table 1: Power communication business characteristics

Category	Application instances	Delay requirement	Business features
Emergent	Dispatching data network,	50∼100 ms	Random
	Power primary service monitoring,		Random
	Dispatching telephone control.		Cyclical
Important	Integrated data network monitoring,	100∼2000 ms	Random
	Power environment monitoring.		Cyclical
Normal	Video conference control,	>2000 ms	Cyclical
	Network topology management.		Random

The communication business related to the dispatch data network and the failure that will cause a power interruption are all emergency business, such as from the failure to the issuance of alarm reminders, dispatcher orders to the communication equipment to complete the routing reversal [17]. Such businesses are sensitive to latency, and if the processing time is prolonged, it may trigger a cascade of power business, resulting in a large accident [18].

At present, the network management system of the power communication network and the subsystems it contains are mainly divided and governed [19–25]. The cost of overall operation and maintenance in this mode is high, so the integrated and integrated system based on cloud platforms has become a more reliable centralized implementation scheme [26,27]. This year, many cloud scheduling algorithms have been proposed, such as document [28], which designs a dual-queue job dispatcher for wind farms, and considers the deadline and priority of jobs to make job scheduling decisions. Document [29] designs a fast convergence and efficient multi-objective game scheduling algorithm for distributed cloud platforms. Document [30] implements business scheduling based on dual-threshold Elastic Resource Management algorithm in high-energy physics.

The rest of the paper is organized as follows. Section 2 introduces the edge IoT agent task scheduling model. In section 3 we introduce our reliable scheduling method for sensitive power business based on deep reinforcement learning. Section 4 is simulation and we make conclusion in section 5.

2 Edge IoT Agent Task Scheduling Model

The edge IoT agent task scheduling model we designed is shown as Fig. 1.

The business deployment model of IoT agent consists of the cloud-based control platform, the Internet-Link-of-Things Agent server at the edge of the network, and various terminal devices of the access system [31]. Terminal devices are responsible for data collection and submission to the Federation of Things Agents. The Federation of Things Control Platform service provides services

such as deployment monitoring, task scheduling and resource management of the Federation of Things Agents cluster, monitors the deployment and operation of the Federation of Things Agents in various regions, manages computing resources, and schedules processing tasks. Due to the different terminal devices managed by the Material Link Agent, the tasks it can handle are also different. The management platform allocates the computing resources of the nodes to the corresponding processing nodes according to the scheduling algorithm according to the type of tasks and the requirements of resources, that is, edge material association agent, and then the nodes perform the specific task. To enable the computing task process in the server to run concurrently, the Docker container technology is introduced to virtualize the computing resources.

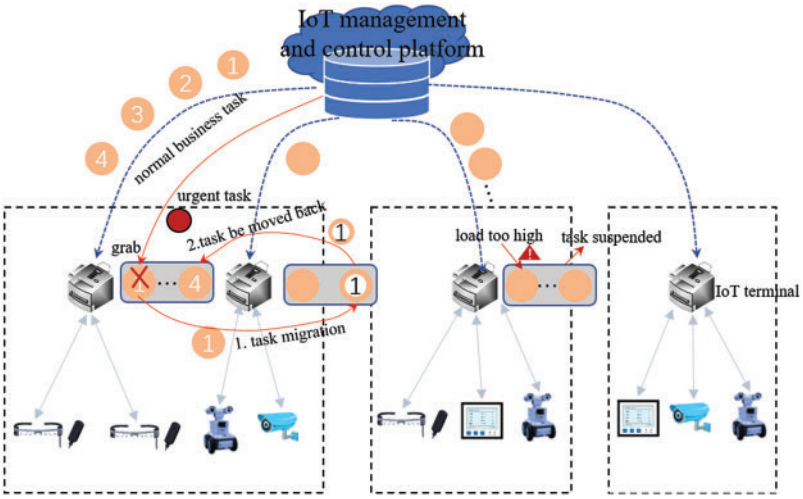


Figure 1: Edge IoT agent task scheduling model

2.1 Network Mode

Tasks are stored in the cache sequence according to their arrival order in the Federation of Things Agent, which is then taken out in a first-in, first-out manner to allocate a corresponding amount of virtual resources. In this article, the unit of computing resources is the number of CPU cycles per second (CPU cycles/s), which represents the computation capability in task concurrency. The amount of virtual computing resources allocated for each task is derived from the deep reinforcement learning algorithm. Based on this amount of resources, the container creation, modification and destruction operations are selected. The specific process is detailed in the virtual resource container model. In the operation of the edge server network, there will be problems such as task preemptive execution, server overall failure or server memory failure, so the corresponding computing tasks need to be migrated to maintain the stable and safe operation of the network.

This paper chooses the open-source container management engine Docker as the environment for the task to run, and designs a virtual resource container running model within the edge server:

1) When a task arrives, the following behaviors occur based on the actions given by the decision model:

a) If an idle container is not destroyed, the service type is the same, and the capacity is appropriate, the task will run in the container.

b) If the free container is not destroyed, but its service type or capacity is not suitable, and the remaining capacity in the server is insufficient, a new container will be created after destroying part of the free container.
c) If there are no free containers, create new containers directly if there is still capacity. Otherwise enter the waiting queue.

2) Start destroying containers in time when tasks are completed and destroy containers if not used within n time slots.

2.2 Task Scheduling Model

Power communication business periodically collects monitoring data from communication devices and networks and submits it to power communication cloud management system. In the event of an unexpected random event, such as a business interruption or equipment failure, it is also handled by the cloud management system [32]. Cloud Platform Business Scheduler schedules business according to business priority. Depending on the size of the power communication network managed by the cloud management system and the level to which the management and control system belongs, the number of businesses and priority levels in the model also differ. The scheduler sets a queue for each priority. Businesses with the same priority enter the same queue. The scheduler sends the business to the management and control system at the appropriate time.

Limited computing resources on cloud platforms result in limited queue length. When business increases dramatically, the number of businesses waiting to be dispatched in the queue increases, and the queue is blocked, resulting in increased business delay [33]. The scheduling algorithm proposed in this paper dynamically adjusts the queue length according to the queue history, calculates the queue congestion degree and converts it into the waiting cost of the whole queue service. By minimizing the total cost through the optimization algorithm, the blocking rate and the service delay of the high-priority service can be effectively reduced.

In the process of regular power business scheduling, there may be some urgent tasks that need to be preempted for scheduling, or the business processing server may have excessive load or operation failure. At this time, in order to ensure the normal and stable operation of the power network, the tasks in the server need to be migrated between containers or servers, and the tasks need to be reallocated after migration [34,35]. To combine task migration with resource allocation, this paper disassembles the task migration process:

1) The IOT control platform sends task execution requests to the edge IOT agents at regular intervals, and the IOT agents feed back the operation of their internal containers to the control platform. If the management and control platform has sudden emergency businesses that need to be dispatched, if the management and control platform has not received feedback from the Internet of things agent for a long time, or the feedback indicates that there is a problem in the operation of a container, task migration should be carried out for the problem.

2) If there is an urgent business that needs to be scheduled, the corresponding IOT agent needs to suspend the tasks being executed on the resources required by the business, record the task execution at this time, and migrate these tasks to other appropriate IOT agents. When the emergency task is completed, the task can be moved back to the original IOT agent to continue execution.

3) If the physical agent itself fails:

a) If there is a problem with the processing server, the control platform selects another processing server as the migration target for the task;

b) If a container fails, the control platform notifies the edge server to collect and package the calculated parts, and judges that it should still be executed in the server or migrated to other servers.

4) After the migration is completed, the task can be regarded as the same as other newly arrived tasks, and enter the task queue to wait for the reallocation of resources.

2.3 Network Delay Model

Given the volatile defects of memory, a memory database recovery mechanism combining replica snapshots and log files is proposed. This mechanism uses the existing memory database replica snapshots and logs to recover the memory database, which can effectively improve the speed and accuracy of data recovery and maintain the integrity of the memory database.

Network delay consists of calculation delay, transmission delay, delivery delay and queuing delay. Since this scenario is basically based on the edge layer, this paper mainly considers the calculation delay and queuing delay within the edge server, as well as the transmission delay and delivery delay between the edge servers. Because the transmission speed of the edge server is faster, the transmission delay can be ignored. The network delay formula can be expressed as:

$$t = t_{com} + t_{lin} + t_{pro} \tag{1}$$

Computing Delay. Calculating latency can be defined as the amount of time it takes a computing task to execute from start to finish, so it is related to the computing power of the edge server and the amount of computing required to compute the task. The formula is:

$$t_{com} = \frac{F}{v_{com}} \tag{2}$$

In the formula, F is the amount of computation for the calculation task, and v_{com} is the virtual computing resource allocated to it by the edge server, that is, the CPU consumption per second.

Queue Delay. Because of the limited computing power of the edge server, some computing tasks cannot get computing resources to be processed at the same time they reach the server and are therefore dumped into the cache, waiting for enough computing resources to be allocated to it. Queuing delay is usually affected by the server's total computing capacity, computing speed, the total task amount arriving at the server before this task, etc. It can not be calculated accurately by the formula. It is necessary to simulate the task execution sequence in the server based on the actual situation, and then deduce the size of queuing delay of a task.

Delivery Delay. When a server fails, it may be necessary to transfer tasks within the failing server to other server. The time incurred in this process is the delivery delay, mainly related to the geographic distance between the two servers and the delivery speed. The formula is:

$$t_{pro} = \frac{d}{v_{pro}} \tag{3}$$

In the formula, t_{pro} is the delivery delay , and d is the geographic distance between the two servers, and the v_{pro} is delivery speed.

2.4 Computing Energy Consumption Models

Service consumption is another performance indicator in the resource allocation optimization algorithm, which represents the energy consumption of edge server system during the whole process.

Service consumption is defined as the sum of computing consumption, transmission consumption and configuration consumption in common network scenarios.

$$e_{it} = e_{op,it} + e_{tran,it} + e_{con,it} \tag{4}$$

Computing Consumption. Computing consumption corresponds to computing latency in response latency, which refers to the consumption that the server incurs in completing the computing task for each computing task assigned to it. The formula is as follows:

$$e_{op,it} = \sum_{j} x_{ijt} f_{ijt} a_{ij} \tag{5}$$

In the t-slot, x_{ijt} means that if the calculation task J is handled by Edge Server i, x_{ijt} is 1, otherwise 0, f_{ijt} means the amount of computing resources that the server assigns to the calculation task, and a_{ij} means the value that the edge server consumes per unit of resources for the calculation task it generates.

Transmission Consumption. Transmission consumption represents the energy consumption incurred during data transfer between edge servers. Transmission consumption corresponds to but does not correspond to the transmission delay in response delay. In the calculation of network delay, the transmission delay is ignored and only considered. In the calculation of transmission consumption, the consumption produced by the two steps of transmission and transmission in network communication is calculated together. The formula is as follows:

$$e_{tran,it} = \sum_{j} y_{ijt} W_{it} q_i + \sum_{j} y_{ijt} D_{it} h_i \tag{6}$$

where y_{ijt} means that server i migrates computing tasks to other server j. The first item of the formula is the consumption generated by the transmission process. W_{it} represents the size of the data sent by the server, and q_i represents the transmission consumption per unit of data volume of the server; The latter item is the consumption generated in the transmission process, D_{it} is the geographical distance between server i and j, and h_i is the transmission consumption per unit distance of server i.

Configuration Consumption. Configuration consumption is incurred by edge servers during container building, container maintenance, and container destruction. The formula is:

$$e_{con,it} = ne_{it} * e_{new} + nm_{it} * e_{mai} + nd_{it} * e_{des} \tag{7}$$

Among them, ne_{it}, nm_{it} and nd_{it} represent the number of containers built, maintained and destroyed by edge server I within time slot t, and e_{new}, e_{mai} and e_{des} represent the energy consumed by these three processes.

3 Reliable Scheduling Method for Sensitive Power Business Based on Deep Reinforcement Learning

By combining the perception of in-depth learning with the decision-making ability of in-depth learning, in-depth reinforcement learning can be controlled directly according to the input knowledge, thus making the solution more similar to human thinking. At present, there are many kinds of reinforcement learning. Common basic algorithms include Deep Q Network (DQN), AC (Actor-Critic) and the DPG (Deterministic Policy Gradient) algorithms. Among them, the DDPG algorithm is gradually formed by the DPG algorithm. The AC algorithm forms the A3C algorithm by adding an asynchronous Actor-Critic network.

In this paper, A3C algorithm is selected to improve among many deep reinforcement learning algorithms to solve the edge network resource allocation problem. The A3C algorithm is based on

the Actor-Critic model and uses agents to learn optimal strategies. Still, the difference is that the A3C algorithm establishes multiple agents to interact with the environment and provides each agent with a copy of the environment so that each agent can interact with its own copy of the environment. These multiple agents are called worker agents and have an independent agent that becomes a global network to which all agents report, a global network that integrates all experience. If only a single agent interacts with the environment, it is possible to make the samples highly relevant, and asynchronous learning can be achieved by using the above methods, thus breaking this correlation.

As mentioned earlier, this paper uses the time-slot method, in which, at most one task arrives or completes a calculation within a time slot. If there are tasks arriving and the server has remaining computing capacity or tasks completing and there are computing tasks in the queue, the current state of the network is formed as state vector S_t, and the corresponding action values are generated through the A3C algorithm. To minimize network latency in scenarios and balance network energy consumption, the objective function of this paper is:

$$\begin{aligned} &\min \sum_{i=1}^{I} \left[\beta \left(t_{i,com} + t_{i,lin} + t_{i,req}\right) + \gamma \left(e_{op,it} + e_{tran,it,} + e_{con,it}\right)\right] \\ &\text{s.t.} \sum_{i}^{I} x_i c_i \leq C \end{aligned} \tag{8}$$

In the formula above, $t_{i,req}$ represents the expected wait delay of a calculation task, β and γ are balance factors to balance the delay with the unit of energy consumption. The above formula indicates that the objective of this article is to minimize the difference between actual and expected delays and network consumption.

Therefore, the reward function r_t value of A3C is set to:

$$r_t = \beta \left(n_{com,t} + n_{lin,t}\right) t_{i,req} + \gamma \left(e_{op,it} + e_{tran,it,} + e_{con,it}\right) \tag{9}$$

where $n_{com,t}$ and $n_{lin,t}$ represent the number of tasks in the server container and queue at time slot t, respectively, and t_{slot} represents the length of the time slot. In the algorithm architecture, the target function of A3C is:

$$\min J(\theta) = E[A(s,a) \log_{\pi}(a|s)] = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} A^{\theta}(s_t, a_t) p_{\theta}(a_t^n | s_t^n) \tag{10}$$

As mentioned above, restrictions indicate that the sum of the computing capacity allocated by the server in the current time slot cannot exceed the total computing capacity of the server. This paper introduces the A3C algorithm to build the objective function, where $A^{\theta}(s_t, a_t)$ represents the dominant function and is the difference between the action value function $Q_{\theta}(s_t, a_t)$ and the expected value function $V_{\theta}(s_t)$:

$$A^{\theta}(s_t, a_t) = Q_{\theta}(s_t, a_t) - V_{\theta}(s_t) \tag{11}$$

Here, $V_{\theta}(s_t)$ can be derived from the Bellman optimization equation:

$$\begin{aligned} V_{\theta}(s_t) &= \max E_{\theta}\{R_t | s_t, a_t\} \\ &= \max \sum_{s_{t+1}} [\Pr(s_{t+1} | s_t) R(s_{t+1}, s_t, a_t) V_{\theta}(s_{t+1})] \\ &= \max \left\{(1-\gamma) R(s_t, a_t) + \gamma \sum\nolimits_{s_{t+1}} \Pr(s_{t+1} | s_t, a_t) V_{\theta}(s_{t+1})\right\} \end{aligned} \tag{12}$$

Similarly, $Q_\theta(s_t, a_t)$ can be derived from this:

$$Q_\theta(s_t, a_t) = R(s_t, a_t) + \gamma \sum_{s_{t+1}} Pr(s_{t+1} | a_t, s_t) \max Q_\theta(s_{t+1}, a_{t+1}) \quad (13)$$

Where γ is the discount factor, $R(s_t, a_t)$ is the reward value generated when action a_t is selected with s_t status, and its calculation formula has been described previously.

The pseudo code of our algorithm is as follow.

Algorithm 1 Reliable scheduling algorithm

Input: Computing Delay $t_{i,com}$, Queue Delay $t_{i,lin}$, Delivery Delay $t_{i,req}$, Computing Consumption $e_{op,it}$, Transmission Consumption $e_{tran,it,}$, Configuration Consumption $e_{con,it}$
Output: scheduling action value
1. Minimize network latency in scenarios and balance network energy consumption according to $\min \sum_{i=1}^{I} \left[\beta \left(t_{i,com} + t_{i,lin} + t_{i,req}\right) + \gamma \left(e_{op,it} + e_{tran,it,} + e_{con,it}\right)\right]$
2. Set reward function as $r_t = \beta \left(n_{com,t} + n_{lin,t}\right) t_{i,req} + \gamma \left(e_{op,it} + e_{tran,it,} + e_{con,it}\right)$
3. Calculate the expected value function $V_\theta(s_t)$ as Eq. (12) and action value function $Q_\theta(s_t, a_t)$ as Eq. (13)
4. Calculate action value with dominant function $A^\theta(s_t, a_t) = Q_\theta(s_t, a_t) - V_\theta(s_t)$
5. return action value

Based on the above objective function, our A3C deep reinforcement learning training algorithm is shown below.

Algorithm 2 A3C deep reinforcement learning training algorithm

Input: A3C neural network structure of common part, corresponding parameter bit θ, w. A3C neural network structure of this thread, corresponding parameters θ', w′, the number of globally shared iteration rounds T, the global maximum number of iterations T_{max}, the maximum length of single iteration time series within a thread T_{local}, the state feature dimension n, the action set A, and the step size α, β, entropy coefficient c, attenuation factor γ, exploration rate ϵ
Output: A3C neural network parameters of common part θ, w
1. Update time series t=1
2. Reset the gradient update amount of Actor and Critical: $d\theta \leftarrow 0, dw \leftarrow 0$
3. From the A3C neural network synchronization parameters of the common part to the neural network of this thread: $\theta' = \theta$, w′=w
4. t_{start}=t, initialization status s_t
5. Based on policy $\pi(a_t | s_t; \theta)$ Select the action at
6. Execute action at to get reward rt and new state s_t+1
7. t←t+1,T←T+1
8. If s_t is in the termination state, or $t - t_{start} == t_{local}$, go to step 9, otherwise go back to step 5
9. Calculate the position s_t of the last time series:
10. for $i \in (t-1, t-2, \ldots t_{start}) i \in (t-1, t-2, \ldots t_{start})$:
    1) Calculate Q (s, i) Q (s, i) at each time: Q (s, i) $r_i + \gamma$ Q(s,i+1)
    2) Cumulative Actor's local gradient update:
    $d\theta \leftarrow d\theta + \nabla\theta' \log\pi\theta'(s_i, a_i)(Q(s,i) - V(S_i, w')) + c\nabla\theta' H(\pi(s_i, \theta'))$

(Continued)

Algorithm 2 Continued

3) Cumulative critical local gradient update:
$dw \leftarrow dw + \partial(Q(s,i) - V(S_i,w'))2\partial w'$

11. Update the model parameters of the global neural network:
$\theta = \theta - \alpha\ d\ \theta$, $w = w - \beta\ dw$
12. If $T > T_{max}$, the algorithm ends and the A3C neural network parameters of the common part are output θ, w. Otherwise, go to step 3

4 Simulation

We use two more classical schemes as comparison algorithms:

First-come-first-service algorithm (FCFS): FCFS is a common queue-based scheduling strategy. Specifically, in the algorithm, we sort according to the time when the request arrives at the edge system. The first arriving request system will first schedule its tasks, and the next one will not start until the entire scheduling of the requested tasks is completed.

Fixed Priority Algorithm (FPSA): When a user requests to reach an edge system, the corresponding scheduling priority is obtained according to the calculation formula. Unlike the DDoc algorithm, the priority is fixed. During the scheduling process, each time we select the request with the highest priority for scheduling, the next request will not be selected until all tasks in the request have been fully scheduled. The significance of this benchmark is that in a dynamic environment, the priority of application requests should also be dynamically adjusted.

We simulated this in an edge computing system with seven heterogeneous edge servers and a remote cloud data center. The capacity of each edge server is randomly selected from 5 to 10. Generally speaking, local data transfer to the cloud requires a long delay, and we set the data transfer rate between the edge servers to be 20 times as high as between the edge servers and the cloud data center. Assuming that the time of the last container configuration on the edge server is 0.5 s, the effect of this ratio on the performance of the algorithm will also be investigated later. Since the cloud data center has pre-loaded all containers, processing tasks in the cloud data center can save associated container configuration costs. In addition, remote clouds typically have more resources than edge servers. So here we set the average processing time of the same task in the remote cloud to 0.75 times that of the edge server.

The results show that the DDoc is significantly better than other benchmarks. Specifically, by default, the number of requests that the algorithm the DDoc meets the application deadline is at least 1.3 times that of the benchmark algorithm. In addition, we performed several simulations to investigate how different settings of various parameters affect the performance of the algorithm (such as container configuration time and edge server capacity).

Fig. 2 demonstrates the performance of the three algorithms on the data generated by the DAG generator. We gradually increased the deadline for all requests from 0.25× of the original value to 2×, while the other parameters remained unchanged. As the deadline increases, the performance of the three algorithms becomes better, and more application requests are scheduled within the deadline. It can also be seen from the data in the diagram that the DDoc algorithm is significantly better than the benchmark algorithm FCFS and better than the algorithm FPSA. At the default setting of 1.0×, the percentage of application requests that the algorithm DDoc meets the deadline is 4.2× and 1.3× for FCFS and FPSA, respectively. The detailed completion rate comparison of the three algorithms is shown in Table 2.

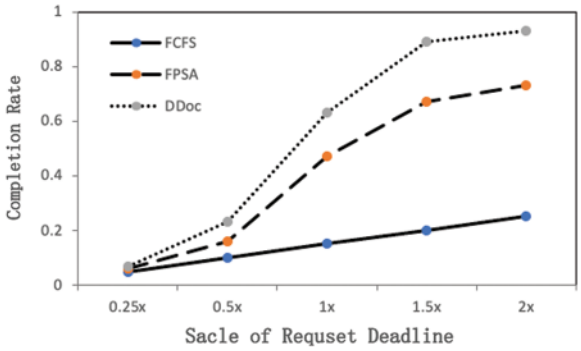


Figure 2: Proportion of algorithms meeting application request deadlines

Table 2: Algorithm comparison in completion rate

Algorithm	0.25×	0.5×	1×	1.5×	2×
DDoc	0.07	0.22	0.63	0.91	0.95
FCFS	0.05	0.1	0.15	0.2	0.26
FPSA	0.06	0.18	0.45	0.64	0.73

To clarify why DDoc is superior to the benchmark algorithm, we further analyze it. We first examine the average task parallelism under each algorithm, which is defined as the average number of tasks executing, container configurations, and so on, occurring each time. Fig. 3 shows that the algorithm DDoc has the highest degree of task parallelism, indicating that DDoc can make better use of server resources than other benchmark algorithms. From the data in the diagram, it can be seen that the task parallelism of FCFS and FPSA is roughly the same and significantly lower than that of DDoc, because during the scheduling process, both FSFSFS and FPSA algorithms are directed at one request and then select another after all their tasks have been scheduled. The parallelism of the algorithms during this period is entirely dependent on the parallelism of the tasks in the request's DAG, and the higher the parallelism of the tasks, the higher the parallelism of the algorithms Consider that in extreme cases, if the DAG degenerates into a linear relationship, the parallelism of the algorithm is 1, meaning that only one task can be operated on at a time. DDoc chooses one of the tasks in a request according to the dynamic priority to operate on each schedule, so it is not limited to the parallelism of DAG tasks, and can improve the parallelism of algorithms by using the relationship between parallel executions among multiple DAGs, which is mainly limited by the number of resources in the system.

Fig. 4 shows the ratio of each algorithm meeting deadlines under different total number of requests in the system. It can be seen that the performance of DDoc algorithm is always better than other algorithms. When the number of requests is small, the performance of each algorithm is not different. However, as the number of requests increases, the proportion of FCFS and FPSA algorithms meeting deadlines decreases dramatically, especially when the number of requests is large. DDoc has always been doing well. The main reason for this is that each time the FCFS algorithm chooses the first arrival request for scheduling, as the number of requests increases gradually, a large number of subsequent arrivals are blocked and difficult to process, and eventually discarded beyond the deadline. While the FPSA algorithm chooses requests to schedule according to priority, it is difficult to make full use of server resources because of the low parallelism of the algorithm in the scheduling process. The

DDoc algorithm adjusts the priority of requests according to the deadline of requests, always chooses the most urgent request to schedule, avoids exceeding the deadline of requests as much as possible, and has the highest degree of task parallelism in the scheduling process, which enables more tasks to be processed simultaneously within the system, maximizes resource utilization, and enables as many requests as possible to be completed within the deadline. However, when the number of requests is large, the performance of the DDoc algorithm is also unsatisfactory, with a completion rate of nearly 50%.

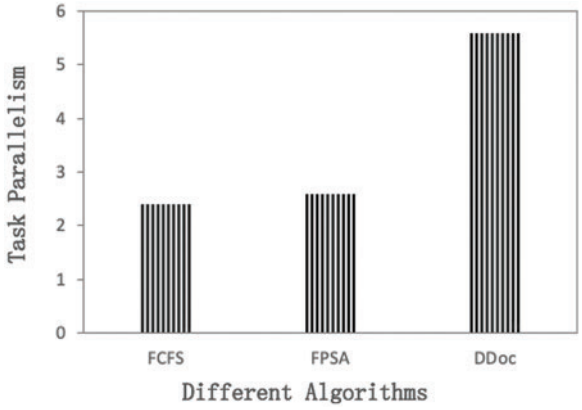


Figure 3: Task parallelism of algorithms

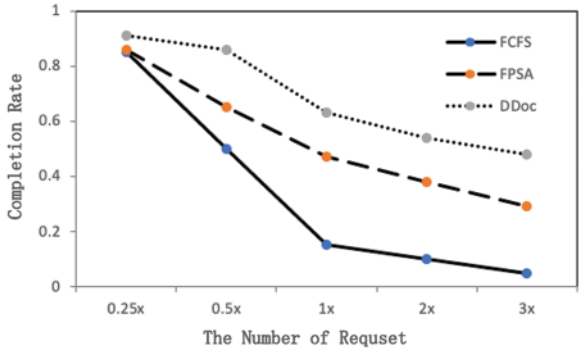
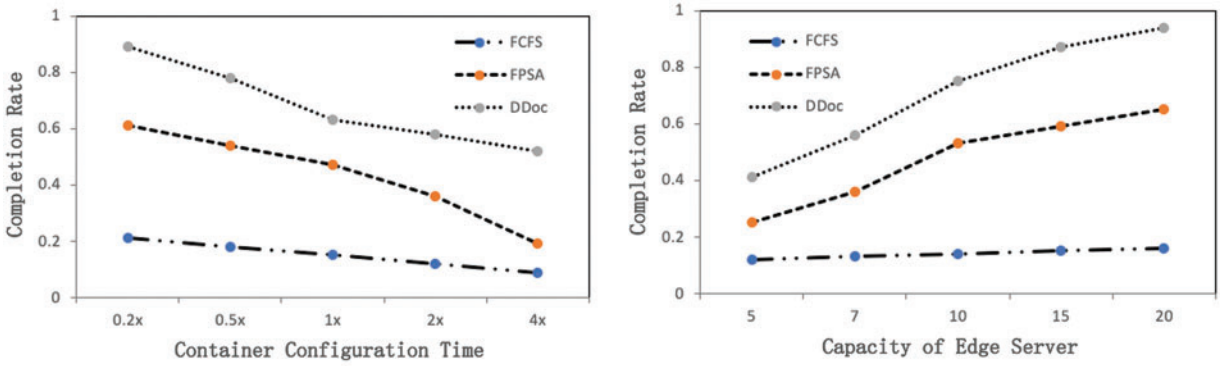


Figure 4: Proportion of algorithm meeting deadlines with different number of requests

We also performed a number of simulations to investigate the effects of different parameter settings (container configuration time, edge server capacity) on the algorithm. Fig. 5a shows that the completion rate of all algorithms decreases with the increase of configuration time. DDoc performs well because the algorithm gives priority to servers that are already configured with the appropriate containers when determining the scheduling scheme for tasks, thus avoiding duplicate configuration of system content servers as much as possible. In addition, Fig. 5b demonstrates the performance changes of all algorithms under different capacity settings. Increasing capacity means that more containers can be configured simultaneously within the system, which also reduces the number of containers to be reconfigured repeatedly and increases task parallelism. From Fig. 3, we have analyzed that DDoc can fully use the parallelism between server and application, so the performance of DDoc will gain significantly as capacity increases. The performance of FCFS is poor and is not sensitive to these parameters, because FCFS always chooses the first arrival request for task scheduling, and when the

first arrival application request requires a lot of processing time, it will continue to block subsequent arrival application requests.



(a) Effect of container configuration time

(b) Effect of edge server capacity

Figure 5: Application request completion rate with different container configuration time and edge server capacity

5 Conclusion

In this paper, the A3C algorithm of deep reinforcement learning network is introduced in network scenarios where heterogeneous edges mostly serve resource collaboration. Task arrival model, task migration model, network latency and energy consumption model are established. Virtualization of edge server computing resources is implemented based on Docker container technology. Task migration and resource allocation are combined to form an optimization mechanism, which can reduce network latency and retain excessive energy consumption. This paper verifies the validity and feasibility of the algorithm by designing a simulation program. It also finds that the algorithm can keep the network running efficiently and adapt to the fast changes of the network when the network is busy. However, there is still some room for improvement in this algorithm, such as the complexity of the algorithm itself is high, so the overall strategy is more complex in operation and learning speed is slightly slower, which should be improved in future research.

Acknowledgement: This paper is supported by the "Research on Digitization and Intelligent Application of Low-Voltage Power Distribution Equipment" [SGSDDK00PDJS2000375].

Funding Statement: This paper is funded by the "Research on Digitization and Intelligent Application of Low-Voltage Power Distribution Equipment" [SGSDDK00PDJS2000375].

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] B. Fan and L. Tang, "Vulnerability analysis of power communication network," *Proceedings of the CSEE*, vol. 34, no. 7, pp. 1191-1197, 2014.

[2] X. Zhou, X. Yang, J. Ma and K. I. -K. Wang, "Energy-efficient smart routing based on link correlation mining for wireless edge computing in IoT," *IEEE Internet of Things Journal*, vol. 9, no. 16, pp. 14988–14997, 2022.
[3] K. Yan and X. Zho, "Chiller faults detection and diagnosis with sensor network and adaptive 1D CNN," *Digital Communications and Networks*, vol. 8, no. 4, pp. 531–539, 2022.
[4] G. Zhou, C. Pan, H. Ren, K. Wang, K. K. Chai *et al.,* "User cooperation for IRS-aided secure MIMO systems," *Intelligent and Converged Networks*, vol. 3, no. 1, pp. 86–102, 2022.
[5] D. Yu, L. Zhang, Q. Luo, X. Cheng, J. Yu *et al.,* "Fast skyline community search in multi-valued networks," *Big Data Mining and Analytics*, vol. 3, no. 3, pp. 171–180, 2020.
[6] F. Wang, G. Li and Y. Wang, "Privacy-aware traffic flow prediction based on multi-party sensor data with zero trust in smart city," *ACM Transactions on Internet Technology*, 2022.
[7] H. Huang, "The design of guangxi power grid provincial-regional integration of telecommunication operation and control system based on cloud platform," Ph.D. dissertation, North China Electric Power University, China, 2017.
[8] X. Zhou, W. Liang, K. Yan, W. M. Li, K. I. Wang *et al.,* "Edge enabled two-stage scheduling based on deep reinforcement learning for Internet of everything," *IEEE Internet of Things Journal*, vol. 10, no. 4, 2023.
[9] W. Liang, Y. Hu, X. Zhou, Y. Pan and K. I. Wang, "Variational few-shot learning for microservice-oriented intrusion detection in distributed industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 8, pp. 5087–5095, 2022.
[10] X. Zhou, X. Xu, W. Liang, Z. Zeng and Z. Yan, "Deep-learning-enhanced multitarget detection for end-edge-cloud surveillance in smart IoT," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12588–12596, 2021.
[11] L. Qi, W. Lin, X. Zhang, W. Dou, X. Xu *et al.,* "A correlation graph based approach for personalized and compatible web APIs recommendation in mobile APP development," *IEEE Transactions on Knowledge and Data Engineering*, 2022.
[12] S. Zeng, G. Chen and F. Qi, "Research on high performance elastic network of cloud data centre," *Computer Engineering and Applications*, vol. 54, no. 7, pp. 89–95, 2018.
[13] X. Zhou, Y. Hu, J. Wu, W. Liang, J. Ma *et al.,* "Distribution bias aware collaborative generative adversarial network for imbalanced deep learning in industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 570–580, 2022.
[14] K. Yan, X. Zhou and J. Chen, "Collaborative deep learning framework on IoT data with bidirectional NLSTM neural networks for energy consumption forecasting," *Journal of Parallel and Distributed Computing*, vol. 163, no. 8, pp. 248–255, 2022.
[15] M. Azrour, J. Mabrouki, A. Guezzaz and Y. Farhaoui, "New enhanced authentication protocol for internet of things," *Big Data Mining and Analytics*, vol. 4, no. 1, pp. 1–9, 2021.
[16] H. Niu, Z. Chu, Z. Zhu and F. Zhou, "Aerial intelligent reflecting surface for secure wireless networks: Secrecy capacity and optimal trajectory strategy," *Intelligent and Converged Networks*, vol. 3, no. 1, pp. 119–133, 2022.
[17] X. Zhou, W. Liang, W. Li, K. Yan, S. Shimizu *et al.,* "Hierarchical adversarial attacks against graph-neural-network-based IoT network intrusion detection system," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9310–9319, 2022.
[18] K. Yan, X. Chen, X. Zhou, Z. Yan and J. Ma, "Physical model informed fault detection and diagnosis of air handling units based on transformer generative adversarial network," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, 2023.
[19] A. K. Sandhu, "Big data with cloud computing: Discussions and challenges," *Big Data Mining and Analytics*, vol. 5, no. 1, pp. 32–40, 2022.
[20] Y. Xu, X. Zhao and Y. -C. Liang, "Robust power control and beamforming in cognitive radio networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 1834–1857, 2015.

[21] Y. Yang, X. Yang, M. Heidari, M. A. Khan, G. Srivastava *et al.,* "ASTREAM: Data-stream-driven scalable anomaly detection with accuracy guarantee in IIoT environment," *IEEE Transactions on Network Science and Engineering*, 2022.
[22] X. Zhou, R. Zhang and C. K. Ho, "Wireless information and power transfer: Architecture design and rate-energy tradeoff," *IEEE Transactions on Communications*, vol. 61, no. 11, pp. 4754–4767, 2013.
[23] Y. Xu, X. Yu, Y. Peng, G. Li and Q. Chen, "Robust energy-efficient power allocation strategy for energy harvesting-aided heterogeneous cellular networks," in *Proc. GlobalSIP*, Anaheim, CA, USA, pp. 803–807, 2018.
[24] L. Dai, B. Wang, Y. Yuan, S. Han, I. Chih-Lin *et al.,* "Non-orthogonal multiple access for 5G: Solutions, challenges, opportunities, and future research trends," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 74–81, 2015.
[25] K. Yan, "Chiller fault detection and diagnosis with anomaly detective generative adversarial network," *Building and Environment*, vol. 201, no. 2, pp. 107982, 2021.
[26] K. Yan, A. Chong and Y. Mo, "Generative adversarial network for fault detection diagnosis of chillers," *Building and Environment*, vol. 172, no. 8, pp. 106698, 2020.
[27] L. Qi, C. Hu, X. Zhang, M. Khosravi, S. Sharma *et al.,* "Privacy-aware data fusion and prediction with spatial-temporal context for smart city industrial environment," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4159–4167, 2021.
[28] Y. Shen, X. Qin and Z. Bao, "Effective multi-objective scheduling strategy of dataflow in cloud," *Journal of Software*, vol. 28, no. 3, pp. 579–597, 2017.
[29] X. Luo, L. Yue and C. Zhen, "Research on job scheduling algorithm on wind farms data center cloud platform based on Hadoop," *Computer Engineering and Applications*, vol. 51, no. 15, pp. 266–270, 2015.
[30] Z. Cheng, H. Li and Q. Huang, "Elastic computing resource management mechanism for highenergy physics cloud platform," *Computer Engineering and Applications*, vol. 53, no. 8, pp. 8–14, 2017.
[31] J. Ren, J. Li, H. Liu and T. Qin, "Task offloading strategy with emergency handling and blockchain security in SDN-empowered and fog-assisted healthcare IoT," *Tsinghua Science and Technology*, vol. 27, no. 4, pp. 760–776, 2022.
[32] T. Zhou, W. Liu, N. Li, X. Yang, Y. Han *et al.,* "Secure scheme for locating disease-causing genes based on multi-key homomorphic encryption," *Tsinghua Science and Technology*, vol. 27, no. 2, pp. 333–343, 2022.
[33] X. Xu, H. Tian, X. Zhang, L. Qi, Q. He *et al.,* "DisCOV: Distributed COVID-19 detection on X-Ray images with edge-cloud collaboration," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1206–1219, 2022.
[34] L. Qi, Y. Yang, X. Zhou, W. Rafique and J. Ma, "Fast anomaly identification based on multiaspect data streams for intelligent intrusion detection toward secure industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 9, pp. 6503–6511, 2022.
[35] C. Hu, W. Fan, E. Zeng, Z. Hang, F. Wang *et al.,* "Digital twin-assisted real-time traffic data prediction method for 5G-enabled internet of vehicles," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2811–2819, 2022.