Delay Optimization for Consensus Communication in Blockchain-Based End-Edge-Cloud Network

Shengcheng Ma[1(✉)], Shuai Wang[1], Wei-Tek Tsai[1,2], and Yaowei Zhang[3]

[1] School of Computer Science and Engineering, Beihang University, No. 37 Xueyuan Road, Beijing, China
mashengcheng@163.com
[2] Digital Society and Blockchain Laboratory, Beihang University, No. 37 Xueyuan Road, Beijing 100191, China
[3] China Mobile Information Security Management and Operation Center, Beijing, China

Abstract. With the rapid development of smart IoT technology, various innovative mobile applications improve many aspects of our daily life. End-edge-cloud collaboration provides data transmission in connecting heterogeneous IoT devices and machines with improvements in high quality of service and capacity. However, the end-edge cloud architecture still remains some challenges including the risks of data privacy and tolerance transmission delay. Blockchain is a promising solution to enable data processing in a secure and efficient way. In this paper, blockchain is considered as an infrastructure of the end-edge-cloud network and the time cost of the PBFT consensus is analyzed from the perspective of the leader's position. Considering the concurrent processing of tasks in cellular networks, multi-intelligent deep reinforcement learning is used to train the assignment strategy of the edge server. The numerical results show that the proposed method can achieve better performance improvement in terms of the time consumption of data processing.

Keywords: End-Edge-Cloud · Blockchain · Delay Optimization · Multi-Agent Deep Reinforcement Learning

1 Introduction

The development of mobile communication technologies fully supports emerging wireless applications. In particular, 5G/6G has facilitated the widespread use of the Internet of Things (IoT). Various applications are based on IoT technologies that provide users with friendly services, such as the Internet of Vehicles (IoV) [1], electronic payments, smart homes, virtual and augmented reality(VR/AR) [2], and unmanned aerial vehicles(UAV) [3]. Massive data will be generated by endpoint devices, processed and stored as valuable information. End-edge-cloud collaboration, integrated with smart IoT devices, edge computing, and the cloud, is considered a promising architecture to cope with a large amount of data [4]. To enhance data security and availability during data communication, blockchain is integrated with end-edge-cloud network. Blockchain is a tamper-proofed and traceable distributed ledger that can prevent the data from tampering or contamination by attackers [6]. The consensus algorithm guarantees reliable data consistency across all nodes of the blockchain [7]. Therefore, blockchain is widely considered a promising technology that integrated in mobile communication [8].

Though great benefits can be gained through the application of cloud-edge-end collaboration, there are still some problems. The tasks can be offloaded to edge servers, while the edge server is not enough to take all the tasks. Sending all data to cloud may cause long propaganda latency, and the latency will degrade system performance [5]. Moreover, blockchain protects data from tampering, but the consensus mechanism leads to more communication and increased data processing delays.

To improve the performance of the end-edge-cloud network, many studies attempt to coordinate tasks at different layers. Article [9,10] exploit the game-theoretical method to design an incentive mechanism to promote the system performance. Resource allocation is an ordinary research orientation to boost system efficiency [11,12]. Yang et al. [13] schedule the computing and network resources effectively and build an intelligence measurement model for healthcare systems. Reinforcement learning, as an optimization method, is often used to solve resource allocation problems. Liao [14] proposes a multi-timescale resource allocation model based on reinforcement learning. The model can optimize physical-layer task offloading in a large timescale. Delay is an important performance indicator. Richard Yu's team [15,16] employs reinforcement learning to solve a resource allocation model for reducing delay in Blockchain-based IoT networks. However, they do not take into account the case of parallel processing of tasks when calculating the processing delay.

The work of [17] builds an edge computing environment among IoT devices with blockchain technologies to support the embedding of security-focused offloading algorithms. Zhang et al. [18] propose a blockchain empowered federated learning framework in digital twin empowered 6G networks. This framework improves the reliability and security of the system and enhances data privacy. Due to the public blockchain having the disadvantages of low transaction throughput, many researchers prefer using permissioned-blockchain in mobile networks. PBFT [19] is a classic consensus algorithm widely used in the permissioned-blockchain. These studies [20,21] attempt to optimize the PBFT consensus algorithm to improve the performance of the blockchain network. Owing to the different volume of resources at each level in the end-edge-cloud network and the algorithmic rules of PBFT, the position of the leader node is also an important factor in the performance of the blockchain networks. Yet, the impact of the position of the leader in the blockchain has not been well investigated in these studies.

The main contributions of our paper can be summarized as follows:

- First, we establish a blockchain-based end-edge-cloud task assignment model based on reinforcement learning. In considering task processing times, we take

into account the situation where tasks are processed in parallel on multiple edge servers. The model aims to reduce task processing latency by assigning tasks to appropriate positions for execution.
- Second, according to the computing, communication, and storage capabilities of layers in the end-edge-cloud architecture, we analyze the time consumption of the PBFT consensus algorithm where the leader deployed on edge and cloud, respectively. The edge server is near the task side but has a small computational power, whereas the cloud has high computational power but the transmission delay is large. Since the leader node broadcasts more communication than the replica node, there is a difference in the total time consumption of the consensus algorithm when the leader is deployed in different locations.
- Third, we implement an optimization method to solve the task assignment problem using the MADDPG algorithm. Our approach uses multi-agent to manage the task assignment for multiple edge servers, and the experimental results demonstrate the superiority of our proposed method.

The rest of this paper is organized as follows: In Sect. 2, we introduce the system model of the blockchain-enabled end-edge-cloud network and analyze the time cost for the PBFT consensus algorithm. Next, we present a reinforcement learning optimization framework to select the appropriate equipment for task processing in Sect. 3. We design a reward function to fit the scenario and implement the framework using the MADDPG method in this section. Then, we provide the experimental and analyze the result to illustrate the improvement of performance. Finally, we conclude this paper in Sect. 5.

2 System Model

To illustrate our system, we first describe the architecture of blockchain-based end-edge-cloud collaboration. Then we build a model to represent the system operation mechanism. Finally, we use the MADDPG deep reinforcement learning method to optimize the policy for blockchain.

2.1 The Architecture of the Blockchain-Enabled End-Edge-Cloud Collaboration

For a typical end-edge-cloud collaboration architecture, there are three layers that constitute the network. As the name is called, those are the device layer, the mobile edge computing layer, and the cloud layer.

For the device layer, it is the endpoint of the architecture and it usually represents the IoT network. In the edge layer, the edge computing servers are usually deployed near the base station. Blockchain service is also supported in the edge layer. The edge computing servers act as the consensus nodes in the blockchain network. The offloading data can be recorded in a block and saved in the blockchain after consensus operation. The cloud layer is deemed as the data

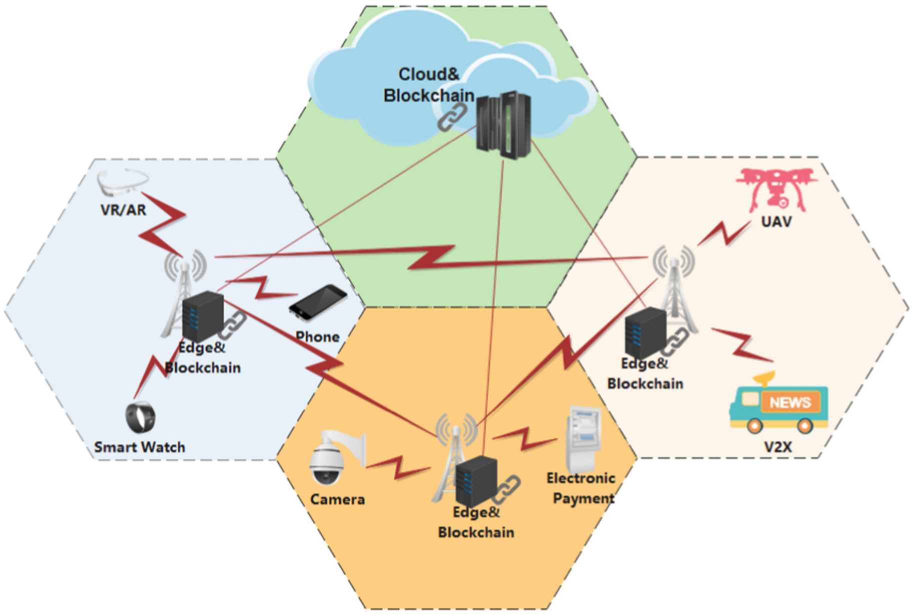

Fig. 1. The Architecture of Blockchain-enabled End-Edge-Cloud Collaboration

center which has sufficient computation, storage, and bandwidth resources. The cloud layer also provides the blockchain service. The cloud server can participate in the consensus communication with the edge servers.

The device layer, the edge layer, and the cloud collaborate with each other and integrate together to consist of a blockchain-based cloud-edge-end architecture. The proposed architecture is shown in Fig. 1.

2.2 Network Model

In the device layer, we assume that a base station covers a cellular network. In each cellular network, there are N active IoT devices connected to the base station. The set of all active IoT devices can be denoted as $\mathcal{D} = \{d_1, d_2, ..., d_N\}$ The ith $(i = 1, 2, ...N)$ active IoT device d_i means that it has a job need to be processed at the time slot t.

In the edge layer, we consider that there are M edge servers represented by the set $\mathcal{E} = \{e_1, e_2, ..., e_M\}$ in the network. Each edge server, like the jth $(j = 1, 2, ..., M)$ edge server, is deployed at the base station and provides a blockchain service. The edge server is responsible to provide service for IoT devices in the cellular network which is covered by the base station. Therefore, we combine the concept of the edge server with the base station, and the jth cellular network also means the network serviced by the edge server e_j. The amount of IoT devices in the whole network is $N \times M$.

We assume that a cloud server C with sufficient resources in the cloud layer, and it is connected to the edge servers by a wired link. All the edge servers and the cloud server constitute a blockchain platform.

The IoT device has lightweight storage and computing capacity, and it only can execute the low load task. A job from the IoT device d_i connected with edge server e_j denotes as $J_{i,j} < s_{i,j}, f_{i,j} >$, and it requires two kinds of resource, $s_{i,j}$ is the data size and $f_{i,j}$ is the required CPU cycles of the job. If the ith IoT device has enough resources to process the job $J_{i,j}$ in time, the job will be handled on the local device. Otherwise, the job will be offloaded to the edge server.

2.3 Wireless Communication

When the IoT device's own resources are insufficient to complete task processing, it will offload the job to the edge server through wireless communication. For wireless communication, the data transmission power on the ith IoT device is represented as P_{I_i,E_j}, and the channel gain from the jth edge server to the ith IoT device is g_{I_i,E_j}. The actual effective transmission data rate is defined as Eq. 1.

$$B_{I_i,E_j} = W \log_2 (1 + \frac{P_{I_i,E_j} g_{I_i,E_j}}{N_0 W}) \tag{1}$$

where W is the bandwidth of the channel, and N_0 is the noise power. Consequently, the speed of task offloading between the ith IoT device to the jth edge server is B_{I_i,E_j}.

2.4 Offloading and Computation

The determinant of where a task will be executed is based on the processing time. Our goal is to minimize the processing time. For the different positions of the job executing, the processing time will be discussed as followed:

Computing on Local Device. For the local computation, the job will be executed immediately without data transferring. The time cost is only generated by the CPU computing, so the time to execute a job on a local device $T_{Local,i}$ can be expressed as

$$T_{Local,i} = \frac{f_{i,j}}{F_i^I}. \tag{2}$$

where F_i^I the CPU cycles per second of the ith IoT device. In addition, we assume that the data size of the job $s_{i,j}$ should be less than the storage capacity of the IoT device S_i^I.

Computing on Edge Server. When a job is too complex, it will be time-consuming to execute on the local device and impact on quality of service (QoS). Therefore, the job will be offloaded to the edge computing server to reduce the processing time. When the job is processed on the edge server, the time cost includes transmission time and computation time.

The transmission time from the ith IoT device to the jth edge server $T^{trans}_{Edge_j,i}$ is represented as

$$T^{trans}_{Edge_j,i} = \frac{s_{i,j}}{B_{I_i,E_j}}. \tag{3}$$

The computation time for job $J_{i,j}$ on the jth edge server $T^{comp}_{Edge_j,i}$ is denoted as

$$T^{comp}_{Edge_j,i} = \frac{f_{i,j}}{F^E_j} \tag{4}$$

where F^E_j is the CPU cycle of the jth edge server. The processing time of $J_{i,j}$ on the jth edge server $T_{Edge_j,i}$ is defined as Eq. (5).

$$T_{Edge_j,i} = T^{trans}_{Edge_j,i} + T^{comp}_{Edge_j,i} = \frac{s_{i,j}}{B_{I_i,E_j}} + \frac{f_{i,j}}{F^E_j} \tag{5}$$

Computing on Cloud Server. If the resource of the edge server is also not enough to process the job, edge server will transfer the job to the cloud server. The processing time of the job in the cloud server consists of three parts. The transmission time from the IoT device to the edge server $T^{trans}_{Edge_j,i}$, the transmission time from the edge server to the cloud server $T^{trans}_{Cloud,j}$, and the computation time on the cloud server $T^{comp}_{Cloud,i}$. The transmission time $T^{trans}_{Edge_j,i}$ is shown as Eq. (3). The transmission time $T^{trans}_{Cloud,i}$ can be represent as:

$$T^{trans}_{Cloud,i} = \frac{s_{i,j}}{B_{Eth}} + T_{route}, \tag{6}$$

where B_{Eth} is the bandwidth of the wire link between the edge server and the cloud server, and T_{route} is the delay generated by routers from the edge server to the cloud server.

The computation time $T^{comp}_{Cloud,i}$ is denoted as:

$$T^{comp}_{Cloud,i} = \frac{f_{i,j}}{F^C}, \tag{7}$$

where F^C is the CPU cycle frequency allocated by the cloud server.

The processing time of job $J_{i,j}$ on the cloud server $T_{Cloud,i}$ is presented as Eq. (8).

$$T_{Cloud,i} = T^{trans}_{Edge_j,i} + T^{trans}_{Cloud,i} + T^{comp}_{Cloud,i} = \frac{s_{i,j}}{B_{I_i,E_j}} + \frac{s_{i,j}}{B_{Eth}} + T_{route} + \frac{f_{i,j}}{F^C} \tag{8}$$

Processing Time of Tasks in Entire Network. For jobs in a cellular network, we assume that a, b, and c are the number of jobs executed on the local device, the edge server, and the cloud server. The range of values of these variables should satisfy $0 \leq a, b, c \leq N$, and $a + b + c = N$. The whole processing time in the mth cellular network can be represented as:

$$T_{cell_j} = \max\{\max_{i \in a}\{T_{Local,i}\}, \sum_{i \in b} T_{Edge_j,i}, \sum_{i \in c} T_{Cloud,i}\}. \tag{9}$$

We calculate the max value of these three parts to support concurrent execution. It means that three layers of servers can process tasks concurrently, which can greatly improve the efficiency of the system. Every IoT device can execute the job respectively, so we use the max function to calculate the processing time of the device layer. In addition, the total amount of data size for jobs offloaded to the edge server should be less than the storage capacity of the edge server. That is $\sum_{i \in b} s_{i,j} \leq S_j^E$. The cloud server is considered to have enough storage capacity to accommodate all offloading jobs, and computing supports parallel processing of offloading jobs from multiple cellular networks.

For all tasks in the network, the total task processing time at the current moment T_{total_t} can be expressed as:

$$T_{total\_t} = \max_{j \in M}\{T_{cell_j}\} \tag{10}$$

2.5 Time Consumption of Blockchain Consensus

After processing, the job should be stored securely. To ensure the traceability and non-tampering of data, this storage process is done by the blockchain system. In this paper, we investigate the PBFT algorithm as the typical blockchain consensus method. In the cloud-edge-end architecture, the blockchain network contains two situations on the basis of the position of the leader node. One is the leader node voted in an edge server, the other one is the leader node in the cloud server. Next, we discuss the time cost in two situations respectively.

Leader Node on the Edge Server. For the case of the leader node on the edge server, we analyzed the time consumption of the different algorithm phases.

In the request phase, a client as the sponsor sends the request of consensus for the $J <i, j>$ to the leader node. The client is an edge server that has completed job J. The transmission from the client to the leader node can be denoted as:

$$T_{req,j}^{trans} = \frac{R + s_{i,j}}{B_{j^c,j^l}} \tag{11}$$

where B_{j^c,j^l} is the bandwidth between the client and the leader node, and R is the REQUEST message except $s_{i,j}$. The REQUEST message includes operation which is the information requiring consensus $s_{i,j}$, timestamp, and client ID. In addition, the client calculates the digest of the request message and signs it. Then, it appends the signature to the plaintext of the REQUEST messages and sends them to the leader node. In the pre-prepare phase, the leader node receives the REQUEST message and validates the signature of the client. Then, the leader composes a PRE-PREPARE message. The PRE-PREPARE message contains the view number, sequence number, and digest of the REQUEST message. The leader node signs the digest of the PRE-PREPARE message and combines the signature with the plaintext and the REQUEST message. Next, the leader multicasts the combination of the PRE-PREPARE message to all the replica nodes. The time cost for multicast can be denoted as:

$$T_{pre-pre,j}^{trans} = \frac{(PP + R + s_{i,j}) \cdot (M-1)}{B_{j^l,j^r}} + \frac{PP + R + s_{i,j}}{B_{Eth}} + T_{route} \tag{12}$$

where PP is the PRE-PREPARE message and the signature, B_{j^l,j^r} is the bandwidth between the leader node and the replica node. There are $M-1$ nodes deployed on the edge servers and one node deployed on the cloud server. The replica nodes receive the message from the leader node and check its correctness. It contains the signature of the leader's PRE-PREPARE message, the signature of the client's REQUEST message, and the digest of the REQUEST message. If this information is right, the replica node accepts the message and turns into the prepare phase.

In the prepare phase, the replica node multicasts the PREPARE message to other nodes. Similar to the PRE-PREPARE message, the PREPARE message also contains the view number, sequence number, and digest. Besides, it adds its node identification to the message. The time cost for transmission can be denoted as:

$$T_{pre,j}^{trans} = \frac{P \cdot (M-1)}{B_{j',j}} + \frac{P}{B_{Eth}} + T_{route} \tag{13}$$

where P is the PREPARE message and the signature, $B_{j',j}$ is the bandwidth between replica nodes. For the special case, a replica node is deployed on the cloud server. The time cost of the transmission for this node can be represented as:

$$T_{pre,c}^{trans} = (\frac{P}{B_{Eth}} + T_{route}) \cdot M \tag{14}$$

Considering the concurrent communications, the total time cost of this phase can be expressed as:

$$T_{pre}^{trans} = \max_{j \in M}(T_{pre,j}^{trans}, T_{pre,c}^{trans}) \tag{15}$$

If each node (including leader and replica) in the blockchain receives at least $2f$ valid PREPARE messages, it will enter the commit phase.

In the commit phase, each node sends the COMMIT message to other nodes. The contents of the COMMIT message are the same as the PREPARE message, so the time cost of the node on the edge server can be denoted as:

$$T_{com,j}^{trans} = \frac{C \cdot (M-1)}{B_{j',j}} + \frac{C}{B_{Eth}} + T_{route} \tag{16}$$

where C is the COMMIT message and the signature, $B_{j',j}$ is the bandwidth between different edge servers. For the node deployed on the cloud server, the time cost can be represented as:

$$T^{trans}_{com,c} = (\frac{C}{B_{Eth}} + T_{route}) \cdot M \tag{17}$$

The total time cost of the commit phase can be expressed as:

$$T^{trans}_{com} = \max_{j \in M}(T^{trans}_{com,j}, T^{trans}_{com,c}). \tag{18}$$

If the number of the COMMIT messages accepted by the consensus node is equal to or greater than $2f+1$, including its own message, the node will send a REPLY message to the client.

In the reply phase, the majority of nodes in the blockchain attain the consensus for a normal situation. Each node will save $s_{i,j}$ on the blockchain and send the REPLY message to the client. The REPLY message contains the view number, timestamp, client ID, node ID, and the result. The time cost of transmission for node deployed on the edge server can be expressed as:

$$T^{trans}_{rep,j} = \frac{RP}{B_{j^r,j^c}} \tag{19}$$

For the node deployed on the cloud server, the time cost can be denoted as:

$$T^{trans}_{rep,c} = \frac{RP}{B_{Eth}} + T_{route} \tag{20}$$

The total time cost of the reply phase can be expressed as:

$$T^{trans}_{rep} = \max_{j \in M}(T^{trans}_{rep,j}, T^{trans}_{rep,c}). \tag{21}$$

The total time cost of the consensus for the leader deployed on the edge server is:

$$\begin{aligned} T_{total\_c} &= T^{trans}_{req,j} + T^{trans}_{pre-pre,j} + T^{trans}_{pre} + T^{trans}_{com} + T^{trans}_{rep} \\ &= \frac{R + s_{i,j}}{B_{j^c,j^l}} + \frac{(PP + R + s_{i,j})(M-1)}{B_{j^l,j^r}} \\ &+ \frac{PP + R + s_{i,j}}{B_{Eth}} + T_{route} + \max_{j \in M}(T^{trans}_{pre,j}, T^{trans}_{pre,c}) \\ &+ \max_{j \in M}(T^{trans}_{com,j}, T^{trans}_{com,c}) + \max_{j \in M}(T^{trans}_{rep,j}, T^{trans}_{rep,c}). \end{aligned} \tag{22}$$

Leader Node on the Cloud Server. In the case of a leader deployed on a cloud server, its consensus communication is different from a leader on an edge server. The difference is mainly concentrated in the transmission path, so the time cost changes accordingly.

In the request phase, the client sends the REQUEST message to the leader on the cloud server. The time cost of the transmission is:

$$T^{trans}_{req,c} = \frac{R + s_{i,j}}{B_{Eth}} + T_{route} \tag{23}$$

In the pre-prepare phase, the leader node sends the PRE-PREPARE message to all replicas on the edge servers. The time cost in this phase can be expressed as:

$$T_{pre-pre,c}^{trans} = (\frac{PP + R + s_{i,j}}{B_{Eth}} + T_{route}) \cdot M \tag{24}$$

In the prepare phase, the replica sends the PREPARE message to other nodes in the blockchain network. The leader node does not participate in message transmission. Therefore, the time cost in this phase can be represented as:

$$T_{pre}^{trans} = \frac{P}{B_{j',j}} \cdot (M-1) + \frac{P}{B_{Eth}} + T_{route} \tag{25}$$

In the commit phase, each node sends the COMMIT message to others. Whatever the leader node is deployed on which position, the communications are identical. Hence, the time cost in this phase is the same as Eq. 18.

In the reply phase, all nodes reply to the client, so the transmission time is the same as Eq. 21.

The total time consumption of consensus for the leader deployed on the cloud server can be denoted as:

$$\begin{aligned} T_{total\_c} &= T_{req,c}^{trans} + T_{pre-pre,c}^{trans} + T_{pre}^{trans} + T_{com}^{trans} + T_{rep}^{trans} \\ &= \frac{R + P + M \cdot PP + (M+1) \cdot s_{i,j}}{B_{Eth}} \\ &+ (M+2) \cdot T_{route} + \frac{P}{B_{j',j}} \cdot (M-1) \\ &+ \max_{j \in M}(T_{com,j}^{trans}, T_{com,c}^{trans}) + \max_{j \in M}(T_{rep,j}^{trans}, T_{rep,c}^{trans}) \end{aligned} \tag{26}$$

2.6 Optimization Model of Minimizing Time for Task Processing and Consensus

In the proposed cloud-edge-end network, we assume that each IoT device has a job to process at a time period. In a time period, all jobs in the network should be completed. Then, the model enters the next stage and devices generate new tasks. Though we suppose the cloud server has adequate resources, the task offloading should satisfy the resource limitation of the edge server. In order to minimize the processing time, the selection of the job processing position needs to be optimized. Combining the above factors, we define the task offloading model as follows:

$$\begin{aligned} &\mathbf{P1}: \min\ T_{total\_t} + T_{total\_c} \\ s.t.\ &C1: \ 0 \leqslant a, b, c \leqslant N \\ &C2: \ a + b + c = N \\ &C3: \ s_{i,j} \leqslant s_i^I, \forall i \in \{1, .., N\} \\ &C4: \ \sum_{i \in b} s_{i,j} \leqslant s_j^E, \forall j \in \{1, .., M\} \\ &C5: \ \sum_{i \in b} B_{I_i,E_j} \leqslant B_{E_j}, \forall i \in \{1, .., M\} \end{aligned} \tag{27}$$

Constraint $C1$ represents that the number of jobs assigned in the end, edge, and cloud should be valid values. Constraint $C2$ represents that all the jobs should be appointed in a position and processed. Constraint $C3$ indicates that the size of the job should be less than the storage capacity of the IoT device that generates the job. Constraint $C4$ indicates that the total data volume of tasks offloaded to the edge server should be less than the storage capacity of the edge server. For IoT devices that offload tasks, constraint $C5$ shows that the total bandwidth allocated to these devices should be less than the total bandwidth of the edge servers.

3 Reinforcement Learning Optimization Framework

The goal of the model is to minimize the task processing time, so we need to assign the tasks to the appropriate servers for processing. Because of Eq. (27) is an NP-hard problem, we cannot obtain a solution using traditional methods within a certain period of time. In addition, each cellular network in the model need an agent, we use Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [22] algorithms to support the parallel processing and solve this problem. For task assignment, we can consider the current network model as the state in RL, and we deem the assignment of tasks as the action. We also need to define a reasonable incentive function as the reward. With the help of the algorithm, the agent completes the position selections for tasks and achieves the goal of minimizing the task processing time.

3.1 State Space

In our model, the cloud-edge-end architecture is constructed by M cellular networks. For a cellular network, we define the environment as the system state $o_j = \{J_j, F_j^I, B_j, S^E, F^E, S^C, F^C\}, j \in [1, M]$. Where $J_j = \{J_{1,j}, J_{2,j}, ..., J_{N,j}\}$ is the set of tasks generated at time slot t in the jth cellular network. For a task $J_{i,j}$, it can be denote as $J_{i,j} = <s_{i,j}, f_{i,j}>, i \in [1, N]$. The element $s_{i,j}, i \in [1, N], j \in (1, M)$ is the data size of task from device d_i in the jth cellular network. Similarly, the element $f_{i,j}, i \in (1, N), j \in (1, M)$ is the required CPU cycles of the task. $F_j^I = \{F_{1,j}^I, F_{2,j}^I, ..., F_{N,j}^I\}$ is the set of computation capacity of the IoT devices. For B_j, it can be written as $B_j = \{B_{I_1,E_j}, B_{I_2,E_j}, ..., B_{I_N,E_j}\}$, and it represents the set of the required bandwidth of tasks which need to be allocated by edge server. S^E, F^E are the storage capacity and CPU frequency of the edge server respectively. S^C, F^C are the resources of the cloud server including storage and calculation. In each time slot, the parameters of tasks generated by the device layer will be stochastic values. For the whole network, we should combine all the cellular networks, so the state space of the environment can be written as S which is $S = \{o_1, o_2, ..., o_M\}$.

3.2 Action Space

The action space is defined as the processing position selection for tasks. The processing position includes local which is IoT device, edge server, and cloud server. In the time slot t, the action $a_j(t), j \in [1, M]$ denotes the selection in the jth cellular network. It can be expressed as $a_j(t) = \{a_{1,j}(t), a_{2,j}(t), ..., a_{N,j}(t)\}$. For $a_{i,j}(t)$, its value represents the selection of processing locations for task $J_{i,j}$. It can be written as $a_{i,j}(t) = 0, 1, 2$, where $a_{i,j}(t) = 0$ means the decision of task processing position is in the local device, while $a_{i,j}(t) = 1$ means the task will be offloaded to the edge server e_j, and the task will be processed by cloud server when $a_{i,j}(t) = 2$. For the entire model, the action space can be represented as $A = \{a_1(t), a_2(t), ..., a_M(t)\}$.

3.3 Reward Function

An efficient incentive function is a key factor for the RL algorithm. When the agent chooses the right action, the reward function should give positive incentives, otherwise the reward function will punish the agent to avoid making wrong decisions. After the action executing in each time slot, the reward function will return a value. The accumulated value is related to the achievement of the optimization objective in the model. Our goal of the proposed architecture network is to minimize the task processing time, so we consider the time cost as criterion in the reward function. Moreover, the action for the processing position selection should be valid. This implies that the resources consumed by the selection must be within the capacity of the servers. Consequently, the reward function for a cellular network can be defined as:

$$\begin{aligned} r_j(t) = {} & \alpha(T + T_{local_t} - 2T_{total_t}) \\ & + \beta(s_j^E - \sum_{i \in (b \cup c)} s_{i,j}) \\ & + \gamma(B_{E_j} - \sum_{i \in (b \cup c)} B_{I_i, E_j}), j \in [1, M] \end{aligned} \tag{28}$$

where T is the time span of the slot t, T_{local_t} is the time cost of processing all tasks on local devices, which means that there is no offloading. T_{total_t} is the time cost after the task assignment of action $a_j(t)$ executing, α, β, and γ are the weight for turning the reward value.

Normally, all the tasks should be completed in the time slot, so $T - T_{total_t}$ will be a positive number. Otherwise, the value of $T - T_{total_t}$ is negative, which is a punishment. Similarly, utilizing edge or cloud computing should enhance the processing efficiency of the task, or else such operations are futile. Therefore, a larger positive difference of $T_{local_t} - T_{total_t}$ indicates more efficient processing and better incentives. This is a good decision. $s_j^E - \sum_{i \in (b \cup c)} s_{i,j}$ denotes that the agent will be punished if the obtained storage space caused by the action is beyond the capacity of the edge server. $(B_{E_j} - \sum_{i \in (b \cup c)} B_{I_i, E_j})$ means that the savings in bandwidth resources will also be rewarded. For the two cases above, $i \in (b \cup c)$ is the same as $a_{i,j}(t) = 1$ or $a_{i,j}(t) = 2$, and they both confine the case in which the task is offloaded to the edge or the cloud server to perform. For the whole system, the reward function should be $R(t) = \{r_1(t), r_2(t), ..., r_M(t)\}$.

3.4 Optimization Solution by MADDPG Algorithm

We apply the MADDPG algorithm to solve the task processing location selection problem. Since our model represents a multiple cellular network scenario, the multi-agent mode of the MADDPG can be effectively compatible with this situation. In addition, the cellular networks share one cloud computing center, so we expect that cellular networks can cooperate rather than compete. The algorithm can well support this situation. By setting the number and policy of adversaries, the cooperative relationship between agents can be realized. To be suitable for multi-agent environments, the basic idea of the MADDPG algorithm is centralized training and decentralized execution, which is the fusion and extension of Deep Deterministic Policy Gradient(DDPG) [23] and Actor-Critic [24] methods. Therefore, we adopt MADDPG to optimize the task assignments in multiple cellular networks. To achieve the goal of minimizing task execution time, we transform the problem into the assignment of tasks in different service locations. We input the state information of the environment into the agents of the RL model. The state information includes the requirements of tasks in multiple cellular networks and the computing resources of each service node in the end-edge-cloud system. In the proposed model, each agent is responsible for task allocation in a cellular network, and multiple agents work together on the selection of task processing locations for the entire network.

In the agent, a DDPG algorithm is constructed with four neural networks: the Actor network, Critic network, Actor Target network, and Critic Target network. The four networks have the same structure, each with three fully connected layers and 64 nodes per layer. The policy in the agent can be defined as μ with parameter θ.

For the Actor network, it generates an action based on the inputting state information of the environment. It can be expressed as $a = \mu_\theta(o|\theta)$. To increase the exploration of the environment, noise is added to the output of the Actor network to form the final Action. The training goal of the Actor network is to maximize the Q value of the accumulated rewards, and the gradient of the expected reward for a cellular network can be denoted as below:

$$\nabla J(\theta) = E[\nabla_\theta \mu_\theta(a|o) \nabla_a Q^\mu(o,a)|_{a=\mu_\theta(o)}]. \tag{29}$$

After the interaction between the action and environment, the agent receives a reward, and the environment transforms into a new state. The quadruple (o, a, r, o') will be stored in the memory buffer. The agent samples a mini-batch of quadruples from the memory buffer, and utilizes experience replay to update parameters of the Critic Target network and Actor Target network. When updating the target networks, a soft update is used in order to avoid too fast parameter changes. The parameters of the Actor Target network and Critic Target network are updated by:

$$\begin{aligned} \theta^{\mu'} &= \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \\ \theta^{Q'} &= \tau\theta^{Q} + (1-\tau)\theta^{Q'} \end{aligned} \tag{30}$$

where $\tau \in (0, 1)$ is the learning rate. The Actor Target network computes the new action $a' = \mu'(o'|\theta^{\mu'})$ and sends the result to the Critic Target network. This action does not require the addition of noise. The Critic Target network adopts the result from the Actor Target network to calculate the target value as below:

$$y = re + \gamma Q'(o', \mu'(o'|\theta^{\mu'})|\theta^{Q'}) \tag{31}$$

For the Critic network, the output is the estimated Q value of the current state and the corresponding action. The training updates θ^Q by minimizing the difference between the action-value function Q^μ and the target value y. The loss function of the Critic network is calculated by:

$$L(\theta^Q) = E[(Q^\mu(o, a|\theta^Q) - y)^2] \tag{32}$$

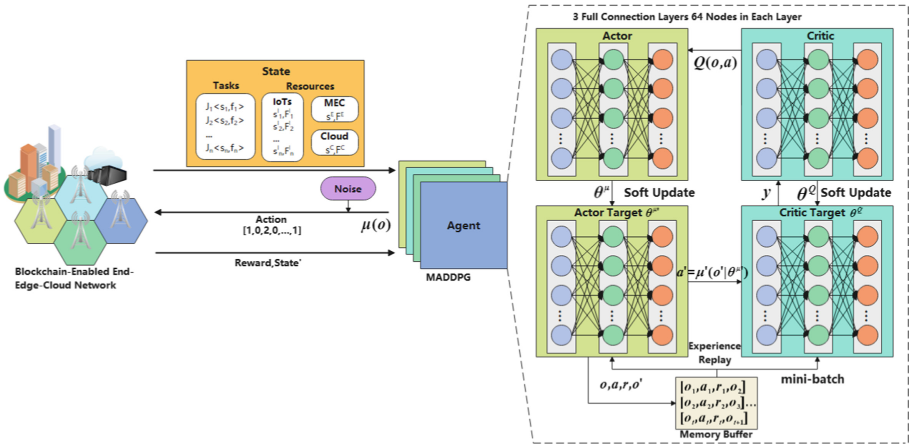

Fig. 2. The MADDPG Model of Task Assignment for Blockchain-enabled End-Edge-Cloud Network

The detail of the task assignment optimization by MADDPG is shown in Fig. 2.

4 Experimental Environment and Results

In this section, we will first introduce our simulation setting. Then, we discuss the performance of our proposed method.

4.1 Simulation Setting

In this simulation, TensorFlow 2.8.0 with Python 3.9 is employed to build the experiment environment, and the OS is Windows 11. We consider an end-edge-cloud network that includes a cloud server and some cellular networks. The number of edge servers can be set as {3, 9, 15}. We set the number of edge servers according to the principle that the number of edge and cloud servers meets the $3f + 1$ relationship required for Byzantine fault tolerance. In each cellular network, the number of IoTs can be set as {10,20,30,40,50}. Each IoT generates a task in slot time. The size of the task is from 0.5 MB to 10 MB, and the required CPU cycles of the task are from 3.5G to 4.5G. The CPU cycle frequency of the IoT device, edge server, and cloud server are 0.5GHz, 2.4GHz, and 20GHz, respectively. Some wireless resource parameters refer to the article [11]. To analyze the consumption of consensus, we assume that both edge servers and cloud servers are blockchain nodes that are involved in consensus communication. For the PBFT algorithm, the message size of R, PP, P, C, and RP is set to 1 KB, and the timeout for consensus was set to ≤ 1 min. In addition, other parameters in the simulation environment are shown in the Table 1.

Table 1. Simulation Parameters

Parameters	Value
Path loss between the IoT and the edge server l_{I_i,E_j}	$140.7+36.7log_{10}(d)$dB
Radio power of the IoT P_i	10 mW
Noise power N_0	-174 dBm
Storage capacity of the edge server S_j^E	100 MB
Total bandwidth of the edge server B_{E_j}	30 MHz
Data rate of wire link from edge to cloud server B_{Eth}	1 GB/s
Delay between the edge and cloud server T_{router}	0.5 s
Slot time T	5 s
Reward value weight α, β, γ	0.5,0.25,0.25

4.2 Result and Discussion

Figure 3 shows the comparison of system reward in our proposed task assignment method with different learning rates. The value of the reward is the average of 1000 training episodes. For learning rates 0.1 and 0.01, the system cannot reach better rewards when it converges. When the learning rate is 0.001, the system can obtain a high reward value and converge quickly. The reward has a negative value because our incentive function has a penalty mechanism, and when tasks are not allocated properly, the reward will have a negative value. Continuing to reduce the learning rate to 0.0001 does not give better results, so we set the learning rate to 0.001 for the following test.

Figure 4 shows the variation of reward with different numbers of edge servers. We selected 3, 9, and 15 as the number of edge servers. These three numbers were chosen because the total number of edge servers plus cloud servers meets the Byzantine requirement of $3f + 1$ for fault tolerance. Each edge server is

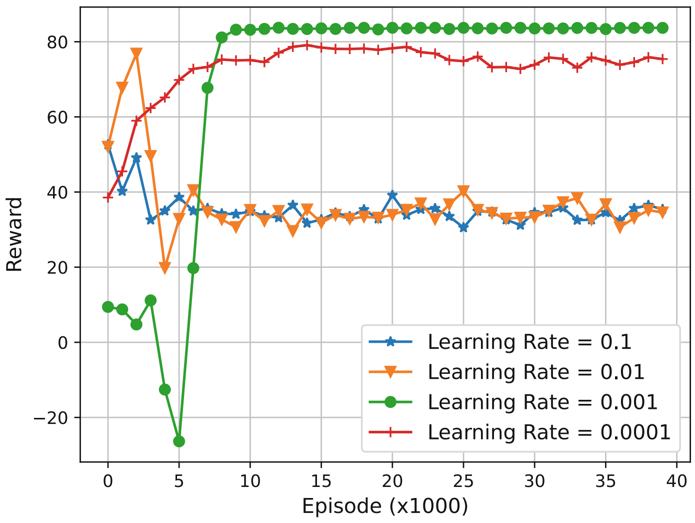

Fig. 3. Reward with Different Learning Rate.

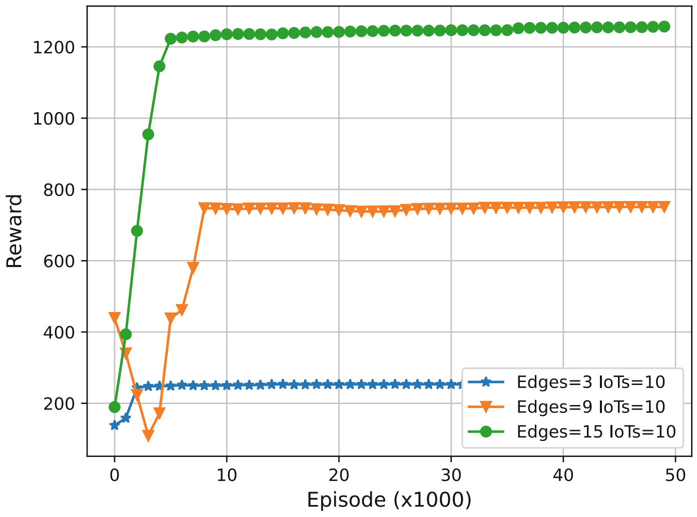

Fig. 4. Reward with Different Numbers of Edge Servers.

responsible for the processing tasks of 10 IoT devices. As the number of edge server grows, our proposed model is still able to consistently guarantee a high reward value. When the number of edge servers is 15, the system's reward figure exceeds 1200. This is because the increase in edge nodes means that more IoT devices are covered and more tasks are handled, so the system gain increases. Each agent can output better decision results for the task, and the model can converge steadily at a high reward value. This validates the stability of the model's decision making capability.

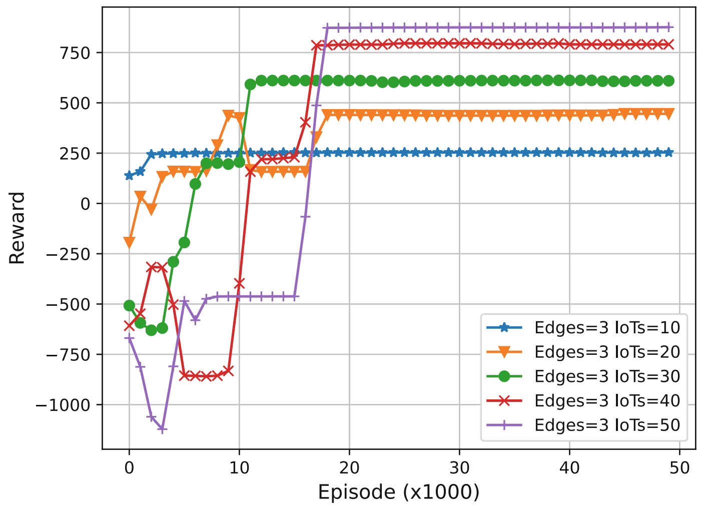

Fig. 5. Reward with Different Numbers of IoT devices.

Figure 5 shows the change in system reward when increasing the number of IoT devices served by the edge server. The number of edge servers is set as 3, and the number of IoT tasks served by each edge node has been increased from 10 to 50. When the number of tasks increases to 50, there are many negative rewards at the beginning of training. This is because our model has a penalty mechanism. Failure to complete all tasks within the current time slot will result in a negative value, and decisions that exceed the capacity of the edge server will also result in a negative reward. As the number of tasks increases, it becomes more difficult for the system to make decisions with constant resources. However, after continuous training of the system, it can eventually converge to a relatively optimal reward value. This illustrates the good adaptability of our method.

Figure 6 shows a comparison of the reward values obtained by the different methods for the case of three edge servers, each handling the task of 10 IoT devices. We compare with some existing schemes, such as the greedy strategy, the random strategy, and similar to the method proposed in the article [15]. As shown in the figure, the rewards of random method do not converge well to higher values and the values fluctuate continuously. For the greedy algorithms, although the reward value has been increasing, the value may be limited to the local optimum without achieving higher gains. Article [15] presents an improved dueling DQN method that is able to converge to a better value, but its convergence speed is slower compared to our proposed method and the convergence value is lower than the value obtained by our method. This demonstrates the advantages of our approach and also illustrates that the adoption of our proposed approach

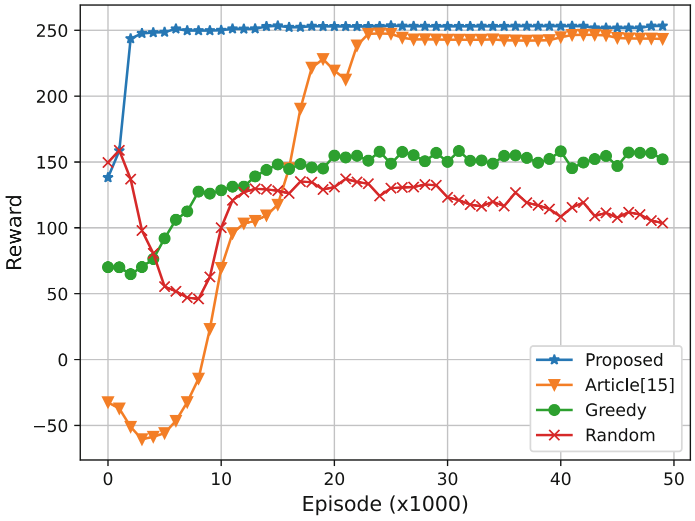

Fig. 6. Reward with Different Method.

allows blockchain networks with end-to-end cloud architecture to process tasks faster and with less transaction latency.

Figure 7 shows the task processing time of different methods under different IoT task. The number of edge servers is set as 3, and the number of IoT tasks for each edge server increased from 10 to 50. We can see that the results of the random method are still inferior to the other three methods. The greedy algorithm has significant time fluctuations due to converging to local optimal values during execution. The method in article [15] is superior to the first two methods, but slightly worse than our proposed method. This demonstrates the advantages of the proposed method.

We also simulated the time consumption of PBFT consensus communication. We conduct comparative experiments on the main factors affecting consensus time, including the number of consensus nodes and the transmission delay of the communication line between edge servers and the cloud server. Figure 8 shows the comparison of time consumption under different consensus node numbers. In this figure, we set the block size to 1MB and the latency between the edge server and cloud server is 0.5 s. In this setting, the consensus time for deploying leader on edge nodes and cloud server is similar and does not differ much. The time consumption of consensus increases with the number of consensus nodes whether the leader node is deployed on the edge server or on the cloud server. When the number of consensus nodes is small, the time consumption of the leader on the edge server is greater than on the cloud server, but as the number of nodes increases, the time consumption is less when the leader is deployed on the edge server than on the cloud server. When the delay is 0.5 s and the number of nodes participating in consensus is 22, the time consumption of consensus exceeds 20 s,

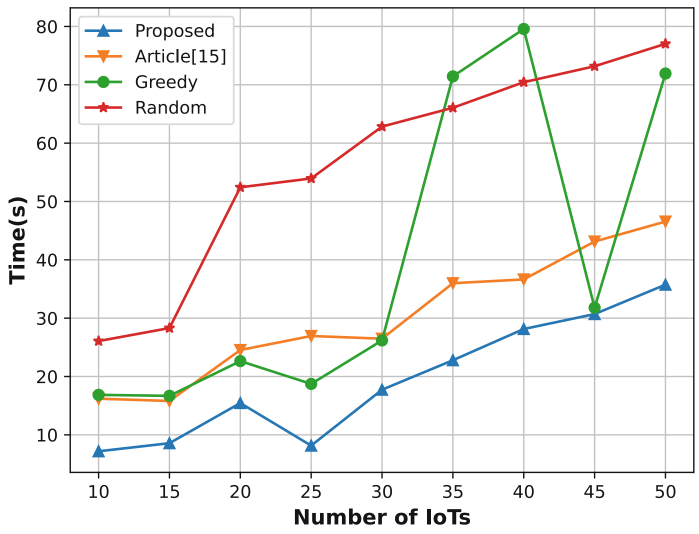

Fig. 7. Task Processing Time with Different Method.

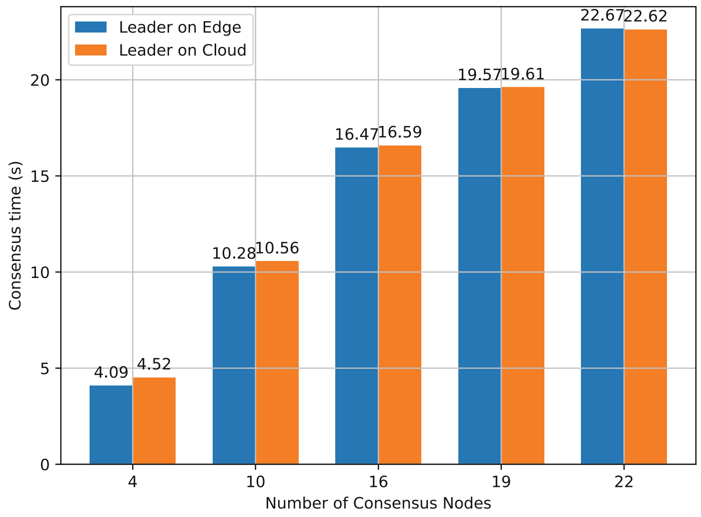

Fig. 8. Comparison of Consensus Time with Different Number of Nodes.

which is hardly acceptable in practical applications. This also indicates that it is inefficient for a large number of nodes to perform consensus in a network with large latency. Figure 9 shows the consensus time of the PBFT algorithm as it varies with the link delay. There are four nodes involved in consensus communication. The delay here mainly refers to the communication delay between the edge server and the cloud server, which is the meaning represented by T_{route}.

From the figure, it can be seen that the consensus time increases with the delay. When the delay is low, the consensus time for the entire blockchain network is smaller when the leader is on the cloud server. As delay increases, the consensus time for the blockchain network is smaller when the leader is on the edge server. Compared to replica nodes, the leader node takes on more communications in the pre-prepare and prepare phases of consensus. The leader node suffers from the delay on the link when it is deployed on the cloud server, so the consensus time increases.

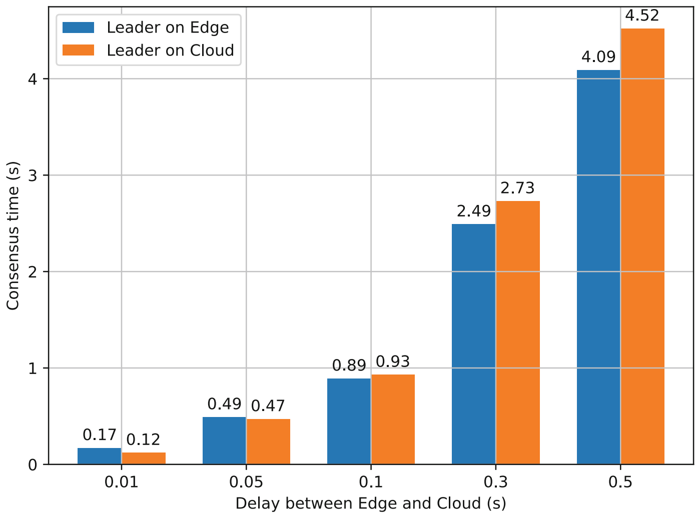

Fig. 9. Comparison of Consensus Time with Different Communication Delay.

5 Conclusions

In this paper, we propose a delay optimized model for the blockchain-based end-edge-cloud network. We considered the situation of multiple network processing tasks in parallel and solved the model using multi-agent reinforcement learning. In addition, we investigate the rule of the PBFT algorithm, and compared the consensus time when leader node is on edge servers and cloud servers. In the experiment, our proposed method can obtain better performance, adaptability, and convergence speed. For the analysis of consensus time, the number of consensus nodes and the delay between the edge and cloud are the main factors. When the delay is large, the leader on the edge is more appropriate. When the number of consensus nodes is large, the leader on the cloud makes the consensus time shorter. With the further development of the research, we will implement a leader generation algorithm that takes into account the resource of the nodes. The algorithm attempts to elect leaders on resource-rich nodes to improve consensus efficiency while ensuring consistency.

Acknowledgements. This work was supported by the National Key R&D Program of China under Grant 2018YFB1402700, and in part by the National Natural Science Foundation of China under Grant 61690202.

References

1. Lu, Y., Huang, X., Zhang, K., Maharjan, S., Zhang, Y.: Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles. IEEE Trans. Veh. Technol. **69**(4), 4298–4311 (2020)
2. Maksymyuk, T., et al.: Blockchain-empowered framework for decentralized network management in 6G. IEEE Commun. Mag. **58**(9), 86–92 (2021)
3. Shubhani, A., Neeraj, K., Sudeep, T.: Blockchain-envisioned UAV communication using 6G networks: open issues, use cases, and future directions. IEEE Internet Things J. **8**(7) (2021)
4. Jiang, M., Wu, T., Wang, Z., Gong, Y., Zhang, L., Liu, R.P.: A multi-intersection vehicular cooperative control based on end-edge-cloud computing. IEEE Trans. Veh. Technol. **71**(3), 2459–2471 (2022)
5. Duan, S., et al.: Distributed artificial intelligence empowered by end-edge-cloud computing: a survey. IEEE Commun. Surv. Tutor. **25**(1), 591–624 (2023)
6. Zhang, S., Wang, Z., Zhou, Z., Wang, Y., Zhang, H., et al.: Blockchain and federated deep reinforcement learning based secure cloud-edge-end collaboration in power IoT. IEEE Wirel. Commun. **29**(2), 84–91 (2022)
7. Mafakheri, B., Heider-Aviet, A., Riggio, R., Goratti, L.: Smart contracts in the 5G roaming architecture: the fusion of blockchain with 5G networks. IEEE Commun. Mag. **59**(3), 77–83 (2021)
8. Li, W., Su, Z., Li, R., Zhang, K., Wang, Y.: Blockchain-based data security for artificial intelligence applications in 6G networks. IEEE Netw. **34**(6), 31–37 (2020)
9. Wang, X., Zhao, Y., Qiu, C., Liu, Z., Nie, J., Leung, V.C.M.: InFEDge: a blockchain-based incentive mechanism in hierarchical federated learning for end-edge-cloud communications. IEEE J. Sel. Areas Commun. **40**(12), 3325–3342 (2022)
10. Ding, Y., Li, K., Liu, C., Li, K.: InFEDGe: a blockchain-based incentive mechanism in hierarchical federated learning for end-edge-cloud communications. IEEE Trans. Parallel Distrib. Syst. **33**(6), 1503–1519 (2022)
11. Feng, J., Yu, F.R., Pei, Q., Du, J., Zhu, L.: Joint optimization of radio and computational resources allocation in blockchain-enabled mobile edge computing systems. IEEE Trans. Wirel. Commun. **19**(6), 4321–4334 (2020)
12. Zhang, X., Peng, M., Yan, S., Sun, Y.: Joint communication and computation resource allocation in fog-based vehicular networks. IEEE Internet Things J. **9**(15), 13195–13208 (2022)
13. Yang, Z., Liang, B., Ji, W.: An intelligent end-edge-cloud architecture for visual IoT-assisted healthcare systems. IEEE Internet Things J. **8**(23), 16779–16786 (2021)
14. Liao, H., Jia, Z., Zhou, Z., Wang, Y., Zhang, H., et al.: Cloud-edge-end collaboration in air-ground integrated power IoT: a semi-distributed learning approach. IEEE Trans. Ind. Inform. **18**(11), 8047–8057 (2022)
15. Li, M., Yu, F.R., Si, P., Wu, W., Zhang, Y.: Resource optimization for delay-tolerant data in blockchain-enabled IoT with edge computing: a deep reinforcement learning approach. IEEE Internet Things J. **7**(10), 9399–9412 (2020)

16. Liu, M., Yu, F.R., Teng, Y., Leung, V.C.M., Song, M.: Performance optimization for blockchain-enabled industrial internet of things (IIoT) systems: a deep reinforcement learning approach. IEEE Trans. Ind. Inform. **15**(6), 3559–3570 (2019)
17. Qu, G., Cui, N., Wu, H., Li, R., Ding, Y.: ChainFL: a simulation platform for joint federated learning and blockchain in edge/cloud computing environments. IEEE Trans. Ind. Inform. **18**(5), 3572–3581 (2022)
18. Lu, Y., Huang, X., Zhang, K., Maharjan, S., Zhang, Y.: Low-latency federated learning and blockchain for edge association in digital twin empowered 6G networks. IEEE Trans. Ind. Inform. **17**(7), 5098–5107 (2021)
19. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, vol. 17, no. 7, pp. 173–186 (1999)
20. Cao, B., Wang, X., Zhang, W., Song, H., Lv, Z.: A many-objective optimization model of industrial internet of things based on private blockchain. IEEE Netw. **34**(5), 78–83 (2020)
21. Chunlin, L., Jing, Z., Xianmin, Y., Luo, Y.: Lightweight blockchain consensus mechanism and storage optimization for resource constrained IoT devices. Inf. Process. Manag. **58**(4), 102602 (2021)
22. Ryan, L., Yi, W., Aviv, T., Jean, H., Pieter, A., Igor, M.: Multi-agent actor-critic for mixed cooperative-competitive environments. In: 31st International Conference on Neural Information Processing Systems (NIPS 2017). Curran Associates Inc., Red Hook (2017)
23. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. In: 4th International Conference on Learning Representations, ICLR 2016 (2016)
24. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans. Syst. Man Cybern. **13**(5), 834–846 (1983)