sensors

MDPI

Article

A Federated Learning Multi-Task Scheduling Mechanism Based on Trusted Computing Sandbox

Hongbin Liu [1], Han Zhou [2,*], Hao Chen [3], Yong Yan [3], Jianping Huang [3], Ao Xiong [2], Shaojie Yang [2], Jiewei Chen [2] and Shaoyong Guo [2]

[1] State Grid Corporation of China , Beijing 100031, China
[2] State Key Laboratory of Network and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
[3] State Grid Zhejiang Electric Power Co., LTD., Hangzhou 310007, China
* Correspondence: bupt_zhouhan@bupt.edu.cn

Abstract: At present, some studies have combined federated learning with blockchain, so that participants can conduct federated learning tasks under decentralized conditions, sharing and aggregating model parameters. However, these schemes do not take into account the trusted supervision of federated learning and the case of malicious node attacks. This paper introduces the concept of a trusted computing sandbox to solve this problem. A federated learning multi-task scheduling mechanism based on a trusted computing sandbox is designed and a decentralized trusted computing sandbox composed of computing resources provided by each participant is constructed as a state channel. The training process of the model is carried out in the channel and the malicious behavior is supervised by the smart contract, ensuring the data privacy of the participant node and the reliability of the calculation during the training process. In addition, considering the resource heterogeneity of participant nodes, the deep reinforcement learning method was used in this paper to solve the resource scheduling optimization problem in the process of constructing the state channel. The proposed algorithm aims to minimize the completion time of the system and improve the efficiency of the system while meeting the requirements of tasks on service quality as much as possible. Experimental results show that the proposed algorithm has better performance than the traditional heuristic algorithm and meta-heuristic algorithm.

Keywords: blockchain; computing sandbox; data privacy; resource scheduling; deep reinforcement learning

Citation: Liu, H.; Zhou, H.; Chen, H.; Yan, Y.; Huang, J.; Xiong, A.; Yang, S.; Chen, J.; Guo, S. A Federated Learning Multi-Task Scheduling Mechanism Based on Trusted Computing Sandbox. *Sensors* **2023**, *23*, 2093. https://doi.org/10.3390/s23042093

Academic Edito: Petros Daras

Received: 15 December 2022
Revised: 7 February 2023
Accepted: 10 February 2023
Published: 13 February 2023

1. Introduction

In the past decade, the global data flow has been growing at an unprecedented speed, and the value behind the data has been paid more and more attention. Intelligent applications supported by massive data will further promote the coordinated development of power enterprises and financial services in the industrial chain. However, due to the problem of data isolation between different departments and systems, the cost of cross-domain data circulation is high, and there is a risk of privacy disclosure, which hinders the full release of the potential value of data.

Traditional centralized financial services in the industrial chain generally have problems such as high cost of verification, incomplete information, difficulty in the supervision of repeated financing, fake financing, and increased financing costs. As a distributed ledger technology [1], blockchain has emerged as a solution to the problem of secure data sharing, providing participants with high-quality data and secure data sharing. At present, some researchers combine industry chain finance with blockchain and integrate blockchain technology into complex business scenarios of industry chain finance. This also brings the problem of the security of the industrial chain financial data privacy in the multi-party

data-sharing environment. Although blockchain-based multi-party data sharing can make the data open and transparent, it cannot guarantee the data privacy security of users. How to protect data privacy in the case of data sharing is a hot topic in current research.

Federated learning (FL), as a new data value-sharing method, aims to solve the problem of data isolation and protect data privacy. Different from traditional machine learning, participants in federated learning do not need to upload their local data but only need to upload parameters and models trained with local data to the central server, and then, the central server will aggregate the global model, thus protecting the privacy and security of their data.

Traditional federated learning relies on a single central server. If this weak central server fails or is attacked, the global model will be inaccurate and even the whole federated learning process will be hindered. In addition, the inability to guarantee whether the central server itself has malicious behavior means that the data privacy and security of each participant will not be guaranteed. The combination of blockchain and federated learning can well protect the privacy of user data and share data value in a decentralized scenario. There have been many studies that use the characteristics of blockchain to replace the central server. At the same time, the nodes with good performance are provided with corresponding rewards; thus, the nodes with better data are encouraged to participate in the training more actively. These mechanisms and schemes use blockchain technology to verify and supervise the original data and final calculated results of federated learning. Although this approach can share the value of data and protect the privacy of users to a certain extent by keeping the data local, it does not take into account whether the computation is credible in the training process of the federated learning model and the privacy leakage caused by malicious node attacks. It ignores the trust supervision in the training and aggregation process of the federated learning model. Therefore, it is a challenging task to build a trusted computing framework that ensures data privacy and security for all participants.

In this paper, to solve this problem, we design a federal learning training supervision mechanism based on state channels by introducing the concept of a computational sandbox in trusted computing. The major contributions to this work are as follows:

- A trusted regulatory framework for federated learning training transactions based on blockchain state channel is designed, and a decentralized trusted computing sandbox composed of computing resources provided by each participant is constructed. The model training process is carried out in the state channel, and the malicious behavior is supervised by a smart contract.
- Aiming at the resource heterogeneity problem of the federated learning participant node, a participant resource management method was proposed to model the resource scheduling optimization problem existing in the process of constructing the state channel. The deep reinforcement learning (DRL) method was used to solve the proposed optimization problem, minimizing the maximum completion time of the system under the condition of meeting the requirements of tasks on service quality.
- We compare the resource-scheduling algorithm based on DRL with the traditional heuristic and meta-heuristic algorithms through simulation experiments. The experiments show that the algorithm has better performance in meeting the requirements of tasks on service quality and reducing the maximum completion time of the system.

The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 introduces the system framework and workflow of the mechanism. In Section 4, the resource-scheduling problem in the process of constructing a state channel is modeled. Section 5 presents the resource scheduling algorithm based on DRL. In Section 6, the proposed algorithm is verified by experimental simulation. Section 7 summarizes the work of the thesis.

2. Related Works

2.1. Decentralized Federated Learning Framework

Federated learning has become increasingly important for modern machine learning in data privacy-sensitive scenarios. The existing federated learning mainly adopts the network topology based on a central server [2–5]. However, in some cases, this connection method is not suitable. For example, there is no central server connecting all users; the communication cost to the central server is unbearable; the central server cannot be fully trusted.

Thus, to further protect data privacy and avoid communication bottlenecks, decentralized architectures have been proposed recently [6–12]. Decentralized architectures remove a central node and each node only communicates with its neighbors (trust each other) by exchanging its local model. Ref. [13] provides a systematic analysis of decentralized learning.

Blockchain is one of the most popular disruptive technologies [14]. With the characteristics of decentralization and data not easily tampered with, it can provide a high degree of guarantee for secure data collection and sharing, paving the way for emerging financial and industrial services.

Many studies have combined blockchain technology with federated learning to ensure the value of sharing data in the context of decentralization and data privacy. Ref. [15] designed a distributed multi-party secure data-sharing architecture based on blockchain authorization to transform the data-sharing problem into a machine learning problem. By sharing the data model instead of revealing the actual data, data privacy was well maintained. To balance the issues of private security and efficiency in fog computing, ref. [16] proposes a new federated learning (FL-Block) scheme based on blockchain, which allows local devices to exchange and update the global model through blockchain. Ref. [17] designed a federated learning system under the blockchain scenario, which uses the reputation mechanism to assist home appliance manufacturers to train machine learning models and predict future consumer demand and consumption behavior. However, these schemes do not guarantee the reliability and security of the training process. The trained model and parameters will be sent to other nodes for aggregation. If other nodes have malicious behaviors, personal privacy information can still be obtained by attacking FL model parameters.

2.2. Resource Scheduling Problem

The existing methods for solving scheduling problems include dynamic programming, probabilistic algorithm, heuristic algorithm, meta-heuristic algorithm, hybrid algorithm, and deep reinforcement learning methods.

The resource-scheduling problem is often abstracted as a target optimization problem. Traditional heuristic resource scheduling algorithms include first-come-first-serve FCFS [18], RR [19], Min–min [20], Max–min [20], etc. Metaheuristic algorithms are algorithms inspired by biological behavior and natural phenomena that imitate biological behavior, including genetic algorithm [21], particle swarm optimization algorithm [22], ant colony algorithm [23], etc. Genetic algorithm [21] applies the principle of biological evolution to obtain high-quality solutions from the search space in polynomial time, and it generates new solutions by randomly modifying better solutions. Ref. [24] proposes a hybridized monarch butterfly optimization algorithm, which is suitable for solving cloud-scheduling problems. Ref. [25] combines two optimization algorithms, namely CS (cuckoo search) and PSO (particle swarm optimization search), to reduce the completion time, cost, and deadline default rate.

In practical application, the cloud system has the following characteristics: (1) the system is large and complex, and cannot be modeled accurately; (2) the timeliness of scheduling decisions requires a high-speed scheduling algorithm; (3) randomness of tasks (or requests), including randomness of task number, arrival time and size. These characteristics pose a challenge to resource-scheduling research in cloud computing. It is difficult

for a particular meta-heuristic or heuristic algorithm to fully adapt to real dynamic cloud computing systems or edge cloud computing systems. Deep reinforcement learning (DRL) is a new method of machine learning, which combines the advantages of deep neural networks and reinforcement learning and has been used to solve resource-scheduling problems of cloud computing in recent years. It has been proved to have strong advantages in many scenarios, especially in complex scenarios of cloud computing [26–31]. Ref. [32] proposed a task-scheduling framework based on Q learning. All requests are prioritized and then tasks are assigned to the virtual machine using a constantly updated policy that minimizes task response time and maximizes CPU utilization. Ref. [33] established a model based on a reinforcement learning K-mean algorithm and developed a physical resource allocation scheme to meet the service quality requirements of users. Ref. [26] proposes a deep reinforcement learning solution based on DRL to effectively solve different cloud resource management problems. The above scheme provides the model and algorithm of the network resource scheduling problem but does not consider the characteristics of federated learning and the training environment.

Aiming at the problems of model attacks of malicious nodes in the federated learning process and whether the computing process is safe and reliable, this paper introduces the concept of computing sandbox to establish the state channel in the federated learning process, ensuring the security and trust of computing in the process of model training and the supervision of malicious behaviors. To solve the container-based resource management problem under this framework, we also propose a resource scheduling strategy for federated learning tasks based on the requirements of tasks on service quality and system completion time, improving the resource utilization rate of network collaborative federated learning and improving the efficiency of the system to complete federated learning tasks.

3. System Model

3.1. System Framework Structure

The system architecture constructed in this paper is shown in Figure 1. By introducing the concept of computational sandbox in trusted computing, a federated learning framework based on state channels is constructed in the blockchain scenario. The model parameter passing and aggregation of federated learning tasks are regulated in the computational sandbox state channel, and the occupied resources are released after the task is completed. The result is then credibly transmitted to the requester. This process supports the trusted calculation and sharing of data, which is invisible to the training participants. The whole system framework is mainly divided into three parts: participant nodes, blockchain and state channel.

The participant nodes can be either the requester or the local training node of the federated learning task. Each participant has user data and resources with training value, representing a certain industry or enterprise, such as financial institutions, credit investigation departments, etc. Participant nodes do not want to share user data with other parties but rather want to share the value of data through federated learning by co-training the global model with local data. The participant node can apply to the blockchain for registration as a training node, and it can apply to cooperate with other participant nodes to conduct federated learning tasks and train the global model.

In the traditional federated learning process, locally trained model parameters need to be sent to the central server or aggregation node, where the global model can be aggregated. In this case, the single point of attack on the aggregation node can easily cause the collapse of the entire federated learning system, local data privacy leakage, and other security problems.

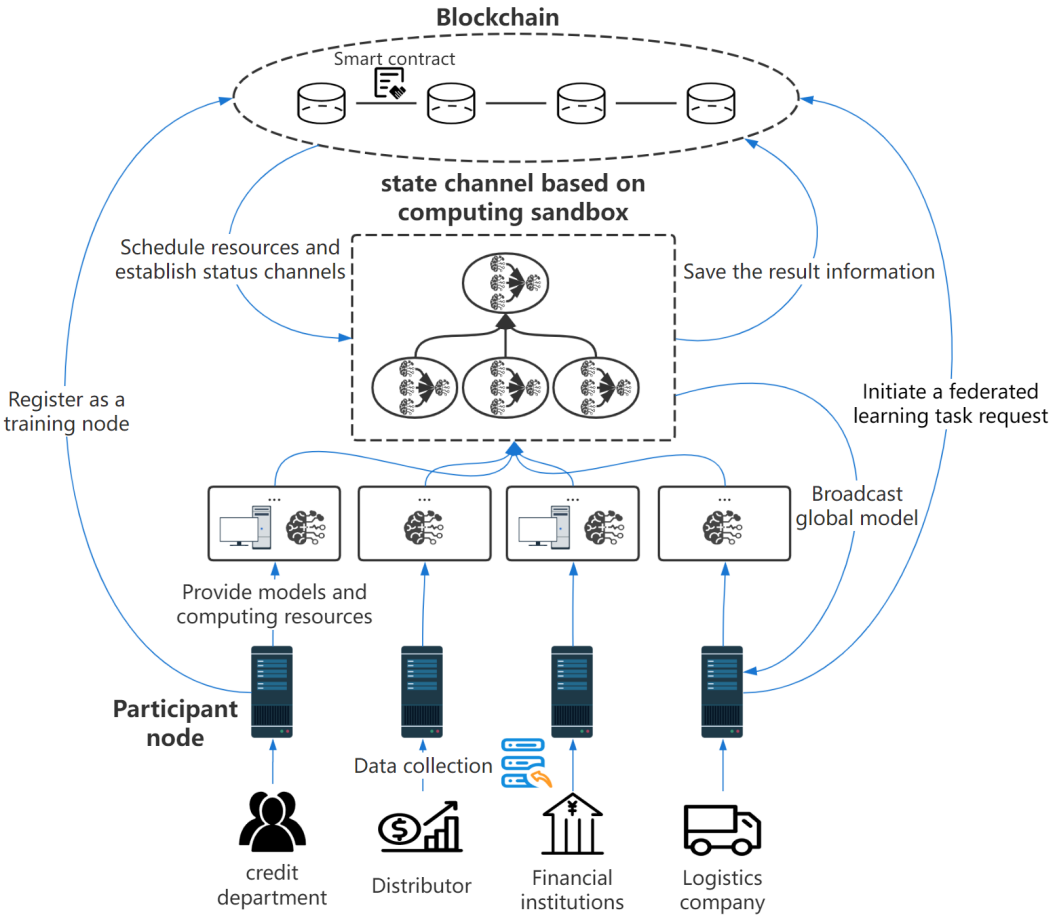


Figure 1. Federated Learning Training Framework Based on State Channel.

To this end, we introduce a computing sandbox as a blockchain state channel. The global model aggregation of federated learning tasks and the calculation of parameters depend on the computing resources and space provided by each participant. We perform the computation in a state channel, which is regulated by the blockchain. In the process of constructing the state channel, the blockchain reasonably schedules resources to the federated learning task according to the requirements of different tasks, making the whole system work more efficiently. The scheduling strategy determines the efficiency and resource utilization of the whole system. Federated learning tasks do not own or retain allocated resources. Instead, the system dynamically allocates them based on current needs using scheduling algorithms to make full use of resources and release used resources immediately after the task is completed. The resource-scheduling problem and scheduling policy are discussed in Sections 4 and 5.

Blockchain is used to manage the status and verification information of each participant, integrate and virtualize various resources registered by each participant, such as computing resources and storage space, and coordinate and supervise the federated learning of each participant through smart contracts. Considering that different participant nodes have different computing power and storage resources, some participant nodes do not have the ability to complete the model training task independently. By integrating the resources of all nodes and constructing state channels with reasonable allocation and scheduling, the problem of resource heterogeneity can be well coordinated and solved. When the task request arrives, the blockchain will conduct resource scheduling, reasonably allocate resources to the task, and establish the federal learning state channel. After that, the training of the federated learning task model and the transmission of parameters will be carried out in the state channel. Any malicious behavior will be supervised by all nodes

and reported to the blockchain to ensure the reliable calculation of data and the security of the model training aggregation process.

To protect the privacy and security of user data, the real physical address of the state channel should not be disclosed to all participant nodes. The blockchain is responsible for masking the real physical address of the state channel from the participant nodes and only providing the virtual address and interface to the participant nodes. The federated learning task is completed in the state channel, and each participant node cannot obtain the model and parameters of other participant nodes, which means the participant node cannot attack the original user data.

3.2. System Work Flow

As shown in Figure 2, the system workflow is mainly divided into the following steps.

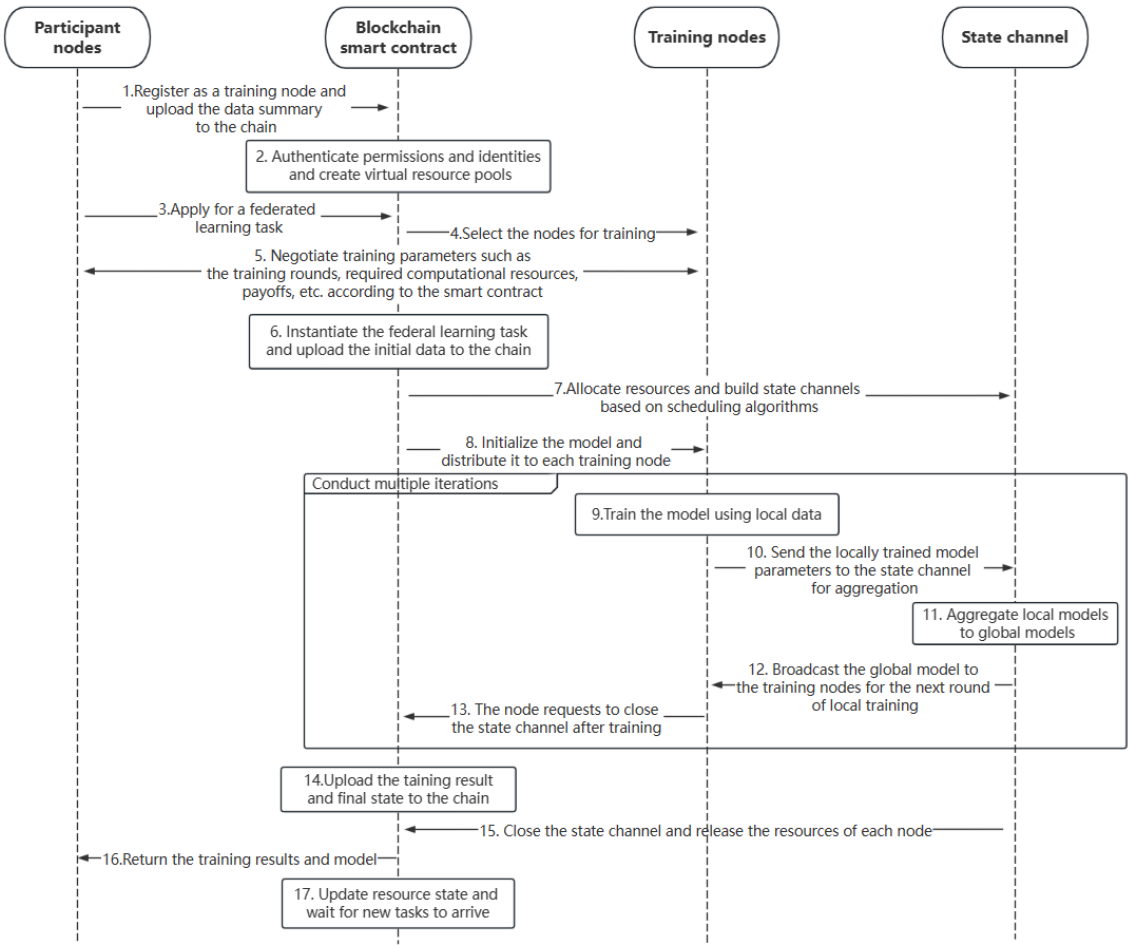


Figure 2. System Working Flow Chart.

In the first step, a participant who wants to instantiate a federated learning task to aggregate the global model and share the value of the data must first register with the blockchain as a training node. Participants submit identity information and describe the data they own and the resources they can provide, including computing unit operation rate, storage space size, cost, etc., for allocation and scheduling by the blockchain.

In the second step, the blockchain authenticates the applied node and integrates the resources provided by the node to establish a virtual resource pool. The blockchain also maintains a virtual resource status table, which records the allocation and usage of resources, the mapping of virtual ports to physical addresses, and so on.

In the third and fourth steps, the participant node applies to instantiate a federated learning training task, and the blockchain selects the training node according to the data description information provided during the registration of each training node.

In the 5th step, each participant node negotiates the relevant parameters of the federated learning training task, including the number of training rounds, the initialization model, the total budget cost, etc., and uses digital signature technology to sign a smart contract.

In the 6th step, the smart contract of blockchain instantiates a federated learning task, determines the initialization priority of the task, saves its initial state on the chain, and places it in the task queue to wait for the allocation and scheduling of computing resources.

In the 7th step, the blockchain reasonably allocates the federated learning tasks in the waiting state to different virtual computing units. The blockchain obtains the physical address of the virtual resource by searching the resource state table and then encrypts it, establishing the trusted computing sandbox as the state channel and updating the resource usage state. The participant node only knows the information of the allocated virtual resources, and it cannot obtain the real physical address of the computational sandbox, so it cannot carry out malicious attacks on the model in training.

In the 8th to 12th steps, participants perform specific federated learning training. The smart contract initializes the model and sends it to each training node. Then, the nodes update the local model with local data and upload the gradient and other parameter information to the state channel for global model aggregation. After the aggregation is completed, the updated global model is sent to each training node for a new round of training until the training is completed. In the training process, the behavior and status of each node are supervised by all nodes. Once malicious behavior is found, it is immediately reported to the blockchain and the malicious node is punished accordingly.

In the 13th to 15th steps, after completing the federated learning training task, the training node updates the final results and status information to the blockchain and then immediately releases the occupied resources, closing the state channel.

In the 16th and 17th steps, the blockchain sends the final result to the task requester, updates the status information of virtual resources, and waits for the arrival of new tasks for resource allocation and scheduling.

In the following sections, we will focus on resource scheduling during the establishment of the state channel and discuss how to allocate resources to make the whole system work more efficiently while satisfying the requirements of tasks on service quality as much as possible.

4. Resource Scheduling Problem Modeling

In the process of constructing state channels for federated learning tasks, different resource scheduling schemes may affect the working efficiency of the system and the quality of completion of tasks. Improper scheduling policies may cause the computing load of some nodes to be too heavy and cause faults, while the resource utilization of other nodes decreases, resulting in resource waste and greatly reducing system efficiency. There may also be unreasonable resource allocation schemes that make it difficult to meet the requirements of tasks on service quality, such as exceeding the budget and the expected completion time. In this section, a model of the resource-scheduling optimization problem under the state channel-based federated learning training supervision mechanism is constructed.

In this paper, the resource scheduling problem is defined as how to assign multiple federated tasks to multiple computing nodes, such as abnormal behavior detection task, risk assessment task, customer behavior analysis task, product intelligence recommendation task, etc., so as to obtain a scheduling scheme that minimizes the completion time of the whole system under the constraints of task cost and task completion time.

We assume that the training node set of the participants is denoted as $Nodes = \{N_1, N_2, \ldots, N_{N_{node}}\}$, where N_i indicates the node i and N_{node} indicates the number of

nodes. Nodes are represented as $N_i = \{PU_1, PU_2, \ldots, PU_{N_{pu}^i}\}$, where $PU_j (j = 1, 2, \ldots, N_{pu})$ represents the j-th processing unit, and N_{pu}^i represents the number of processing units of the i-th node. The processing unit is represented as $PU = \{SIDP, E, COST, TD\}$, where $SIDP$ is the number of the processing unit, E is the execution capacity of the processing unit, $COST$ is the cost that needs to be paid for using the processing unit to execute the unit time, and TD is the communication delay between the node where PU is located and other nodes. $TD_{i,j}$ indicates the communication delay from node i to node j.

We assume that the federated learning task set is expressed as $Tasks = \{T_1, T_2, \ldots, T_{N_{task}}\}$, where T_i represents task i and N_{task} represents the number of tasks. Tasks are represented as $T = \{SIDT, WorkLoad, MaxT, MaxC, PI, N_{train}\}$, where $SIDT$ is the serial number of the learning task, $WorkLoad$ is on behalf of its quota, $MaxT$ represents the maximum completion time a task can bear, and $MaxC$ represents the maximum cost a task can bear. In this paper, $MaxT$ and $MaxC$ are taken as indicators of the requirements of tasks on service quality, PI represents the priority of the task, and N_{train} represents the set of training nodes participating in the task.

Therefore, we can use a matrix to represent the task resource allocation scheme. The allocation matrix is defined as follows:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N_{pu}} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N_{pu}} \\ \cdots & \cdots & \cdots & \cdots \\ x_{N_{task},1} & x_{N_{task},2} & \cdots & x_{N_{task}}, N_{pu} \end{bmatrix} \tag{1}$$

where X is the task resource allocation matrix with the size of $N_{task} * N_{pu}$, and $x_{i,j}$ represents the assignment of the i-th task on the j-th block processing unit, which is defined as

$$x_{i,j} = \begin{cases} 1, \text{ if } T_i \text{ is allocated to } PU_j \\ 0, \text{ if } T_i \text{ is not allocated to } PU_j \end{cases} \tag{2}$$

According to the task resource allocation matrix, we can calculate the completion time of each task. The calculation formula is as follows:

$$ECT_i = \frac{WorkLoad_i}{\sum_{j=1}^{Npu} x_{i,j^*} E_j} \tag{3}$$

Similarly, the cost of each task can be calculated by the following formula:

$$TCost_i = \sum_{j=1}^{N_{pu}} ECT_i * COST_j * x_{i,j} \tag{4}$$

We can also calculate the maximum completion time of the system by the following formula:

$$= MAX_{j \in [1, N_{pu}]} \left\{ \sum_{i=1}^{N_{task}} ECT_i * x_{i,j} \right\} \tag{5}$$

Finally, resource scheduling is abstracted as a goal optimization problem, that is, solving the resource allocation matrix X meets the following conditions:

$$\begin{gathered} \text{Minimize } MAX_{j \in [1, N_{pu}]} \left\{ \sum_{i=1}^{N_{task}} ECT_i * x_{i,j} \right\} \\ \text{s.t. } \forall i \in [1, N_{task}], ECT_i \leq MaxT_i \text{ and } TCost_i \leq MaxC_i \end{gathered} \tag{6}$$

The specific system parameters in this section are shown in Table 1.

Table 1. System Parameters.

Parameters	Meaning
$Nodes$	Set of participant nodes
N_{node}	Number of participating nodes
PU	Processing unit
N_{pu}	Number of processing units
$SIDP$	The serial number of processing unit
E	The execution capacity of a processing unit in instructions per second
$COST$	The execution cost per unit of time of the processing unit
$TD_{i,j}$	Communication delay from node i to node j
$Tasks$	Federated learning task set
N_{task}	The number of tasks
$SIDT$	The serial number of the task
$WorkLoad$	The workload of a task, expressed as the number of instructions
$MaxT$	The maximum completion time the task can tolerate
$MaxC$	The maximum cost that the task can bear
PI	The priority of the task
N_{train}	Set of training nodes for the task
$x_{i,j}$	The distribution of the ith task on the jth processing unit
ECT_i	Completion time of task i
$TCost_i$	The cost of task i
$maxMakespan$	Maximum system completion time

5. DRL-Based Resource Scheduling Algorithm

5.1. Algorithm Framework and Mechanism

In the complex federated learning training environment based on blockchain, the resource-scheduling algorithm needs to select the optimal resource allocation scheme according to the current node resource state and the federated learning task waiting for resource allocation. In this paper, an Actor–Critic resource-scheduling algorithm based on DRL has been designed. The algorithm framework is shown in Figure 3, which is mainly divided into two parts, environment and agent.

The environment is responsible for monitoring the state of each node, managing the computing resources they own, and maintaining a list of tasks awaiting resource allocation. The environment provides the current state information to the agent, including the state of the current resource and the state of the task to be assigned, and it returns the reward of the current operation and the state of the next time according to the actions made by the agent.

The agent is a resource scheduler, making decisions and choosing actions based on the state given by the environment. The system allocates processing resources to tasks based on actions. The agent is composed of the Critic value network and the Actor policy network. The Critic network can score and evaluate the current state, while the Actor network selects the current optimal action according to the environment state.

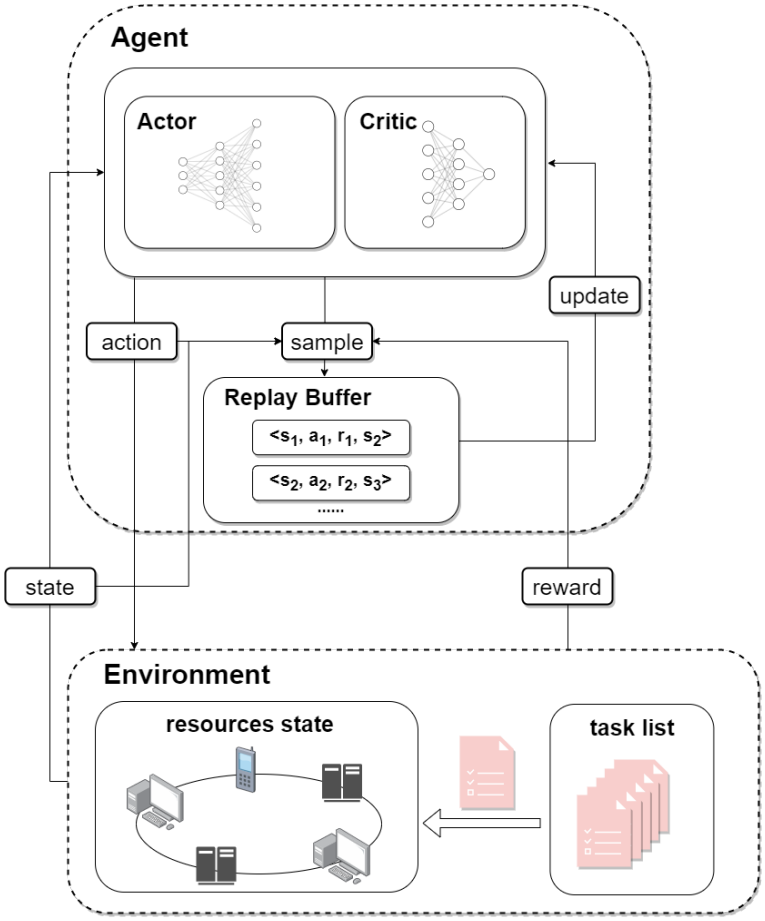


Figure 3. Resource-scheduling algorithm framework based on DRL.

Traditional Q learning has some disadvantages such as a waste of previous experience and correlation of parameter updating. This paper adopted the method of experience replay, set the experience buffer pool R, and stored the environmental state, the action performed, the reward received, and the state of the next moment as experience samples in the buffer pool. When training network parameters, small batch data can be randomly extracted for training and parameter update so as to make better use of the previous experience and break the correlation of parameter update.

Before resource scheduling, the system sorts the tasks waiting for resource allocation based on their priorities and then allocates resources to them. Suppose that the priority of each task is denoted as $Priorities = \{PI_1, PI_2, \ldots, PI_{N_{task}}\}$, where PI_i represents the priority of task i and it is represented as $PI_i = initialPr_i + waitTime_i$, where $initialPr_i$ is the initial priority of the task and related to the $WorkLoad_i$. The lower the workload, the higher the priority, which takes advantage of the short-job-first strategy and helps to reduce the average wait time. $waitTime_i$ is the time the ith task is waiting for resource allocation, which is a dynamic value. The longer the waiting time, the higher the priority, which is conducive to reducing the problem of operation hunger.

When training the Actor–Critic algorithm network, the system makes an action a_t based on the policy network according to the current environment state s_t. Then, the reward r_t is obtained by interacting with the environment, and the new environment state s_{t+1} is obtained, and then, (s_t, a_t, r_t, s_{t+1}) is put into the buffer pool R as a sample. Finally, N groups of data samples were randomly selected from the buffer pool, and the Actor–Critic network parameters were updated by calculating the average value of the gradient.

5.1.1. Train Critic Value Network

Define the discount return U_t as follows:

$$U_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n} \tag{7}$$

where U_t represents the cumulative reward since time t, r_t represents the reward at time t, $\gamma \in (0, 1)$ is the parameter, and the subsequent reward should have a lower weight than the current reward.

In order to judge whether an action is good or bad in the current state, the action value function is defined as follows:

$$Q_\pi(s_t, a_t) = E[U_t \mid S_t = s_t, A_t = a_t] \tag{8}$$

where $Q_\pi(s_t, a_t)$ represents the expectation of discount return U_t, and s_t and a_t are the state at time t and the actions made under this state respectively.

In order to further judge the quality of a certain state, we define the state value function as follows:

$$V_\pi(s_t) = E_A[Q_\pi(s_t, A)] \tag{9}$$

where $V_\pi(s_t)$ represents the expectation of action value function to action A, namely the score of the current state s_t. In this paper, a neural network $v(s; w)$ is used to approximate the state value function. The purpose of the training value network is to make the system score the state more and more accurately, closer to the return given by the real environment.

In this paper, we use the sum of the reward given by the current action and the rating of the state at the next time as an estimate of the true reward, which is defined as follows:

$$y_t = r_t + \gamma v(s_{t+1}; w) \tag{10}$$

where y_t is the estimate of U_t and the target value for training the neural network. Therefore, the loss function can be expressed as:

$$loss = \frac{1}{2}(v(s_t; w) - y_t)^2 \tag{11}$$

So, the gradient of the value network is:

$$g_w = \delta_t \cdot \frac{\partial v(s_t; w)}{\partial w} \tag{12}$$

where δ_t is represented as:

$$\delta_t = v(s_t; w) - y_t \tag{13}$$

Since $v(s_t; w)$ is not the actual observed return, but the estimate of the real discount return, which is often prone to bias. Updating the value network parameters through bootstrapping is prone to overestimation, which is aggravated by continuing to use the already overestimated network to predict the next training values. Therefore, we use target value network $v(s_t; w')$ for estimation. The $v(s_t; w')$ and $v(s_t; w)$ have the same structure, and the same initial value, but different updating methods. Each time y_t is calculated, the $v(s_t; w')$ is used for estimation. The calculation formula is as follows.

$$y_t = r_t + \gamma v\left(s_{t+1}; w'\right) \tag{14}$$

Other calculations remain the same, and the following formula is used when w' is updated.

$$w' \leftarrow \tau \cdot w + (1 - \tau) \cdot w' \tag{15}$$

where $\tau \in (0, 1)$ is the parameter.

5.1.2. Train Actor Policy Network

The policy function $\pi(a \mid s)$ can calculate the probability distribution of the action according to the current state, which is used to make decisions according to the environmental state. This paper uses a neural network $\pi(a \mid s;\theta)$ to approximate the policy function. The goal of training strategy networks is to obtain a bigger discount on the output of actions in the face of different state inputs.

The traditional calculation method of policy gradient is:

$$g_\theta = \nabla_\theta \log \pi(a_t \mid s_t;\theta) U_t \tag{16}$$

In order to improve the convergence efficiency of reinforcement learning algorithm, this paper adopts the policy gradient calculation method based on a baseline, and the gradient calculation formula becomes:

$$g_\theta = \nabla_\theta \log \pi(a_t \mid s_t;\theta)[U_t - b] \tag{17}$$

where b is the parameter independent of action a_t. By subtracting b from U_t, the variance of the gradient is reduced to accelerate the convergence rate of the algorithm. For b, we choose the state value network $v(s_t;w)$ and use Equation (14) to estimate U_t, and we finally obtain the calculation method of policy gradient based on the baseline:

$$g_\theta = -\nabla_\theta \log \pi(a_t \mid s_t;\theta)\delta_t \tag{18}$$

5.2. MDP Model

To use the deep reinforcement learning method to solve the resource-scheduling optimization problem proposed in Section 4, we expressed the problem as an MDP model, which mainly includes state space, action space, and reward. The specific design is as follows.

The state space should include the allocation of each processing resource at time t and the specific state of the task to be allocated at time t, such as the workload, budget, and the maximum completion time of the task that can be tolerated. Therefore, the state space is defined as:

$$s_t = \{X_t, WorkLoad_i, MaxT_i, MaxC_i\} \tag{19}$$

where $WorkLoad_i$, $MaxT_i$ and $MaxC_i$, respectively, represent the workload, maximum completion time that can be tolerated, and the maximum budget cost of the i-th task.

The action space represents the corresponding actions that the agent can perform in the state s_t. The system will remove the task with the highest priority from the wait queue and allocate resources to it. The actions are defined as follows:

$$a_t = \left\{\gamma_1, \gamma_2, \ldots, \gamma_{N_{pu}}\right\} \tag{20}$$

where $\gamma_i \in \{0, 1\}$ indicates how the task is allocated on the i-th resource: 0 for unallocated and 1 for allocated.

The reward is the score that the agent obtains after taking an action based on the current state of the environment to evaluate the goodness of the action. The system calculates the reward of the action according to whether the requirements of tasks on service quality are satisfied after resource allocation and the completion time of the system. With the maximum completion time of the system as the standard of reward, if the action of resource allocation causes the completion time and cost of the task to be greater than the maximum that the task can endure, there will be a huge penalty. The reward can be defined as:

$$r_t = \begin{cases} -5, ECT_i > MaxT_i \text{ or } Tcost_i > MaxC_i \\ -\varepsilon(maxMakespan_t - maxMakespan_{t-1}), \text{ else} \end{cases} \tag{21}$$

where ε is the system parameter.

5.3. Proposed Algorithm

The specific steps of the algorithm are shown in Algorithm 1. The algorithm is trained for M epochs, and the network parameters are updated T times in each epoch. At the beginning of each epoch, a task waiting queue is randomly initialized, and a random initial state is obtained by observing the environment. Before each parameter update, the system obtains a set of (s_t, a_t, r_t, s_{t+1}) samples from the environment and puts them into the buffer pool R. When the amount of data in R reaches the set value, N groups of samples are randomly selected from R for batch gradient update of network parameters.

Algorithm 1 A resource-scheduling algorithm based on Actor–Critic.

Input: Task queues;
Output: A resource-scheduling model;

Initialize the Actor network $\pi(a \mid s; \theta)$.
Initialize the Critic network $v(s_t; w)$ and target value network $v(s_t; w')$.
Initialize the experience replay buffer pool R.
for all $j = 1, 2, \cdots, M$ **do**
  Randomly initialize a task-waiting queue and observe an initial random state s_t.
  for all $i = 1, 2, \cdots, T$ **do**
    Select an action a_t based on the Actor network $\pi(a_t \mid s_t; \theta)$.
    Obtain the environment state of the next moment s_{t+1} and calculate the corresponding reward of the action a_t according to Equation 21.
    The multiple sets of samples (s_t, a_t, r_t, s_{t+1}) obtained by repeating the above process several times are put into the experience replay buffer pool R.
    If the amount of data in R reaches the set value, N batches of samples (s_i, a_i, r_i, s_{i+1}) are randomly selected from R to train the parameters in the neural network.
    Calculate y_i according to Equation 14.
    Calculate δ_i according to Equation 13.
    Calculate the policy network gradient: $g_\theta = -\frac{1}{N} \sum_i \nabla_\theta \log \pi(a_i \mid s_i; \theta) \delta_i$
    Calculate the value network gradient: $g_w = \frac{1}{N} \sum_i \delta_i \cdot \frac{\partial v(s_i; w)}{\partial w}$
    Update policy network: $\theta \leftarrow \theta + \beta \cdot g_\theta$
    Update value network: $w \leftarrow w - \alpha \cdot g_w$
    After iteration k times, update the target value network according to Equation 15.
  end for
end for

In the gradient update process, y_i and δ_i are calculated using Equations (13) and (14). Then, the gradient of the policy network is calculated by weighted average according to Equation (18). y_i served as the target value label of the value network, and the gradient of the value network is calculated by weighted average according to Equation (12). The calculated gradient is used to update the parameters of the policy network and value network, respectively. Finally, the parameters of the target value network are updated after iteration k times.

6. Experiments and Results

6.1. Experimental Settings

To simulate and verify the effectiveness of the proposed deep reinforcement learning algorithm in solving resource-scheduling problems, this paper uses the *Gym* framework of *OpenAI* to model the environment involved in resource-scheduling problems, and it uses the open source framework *Tianshou* [34] based on PyTorch to conduct simulation experiments on the proposed algorithm. The experiment simulates a scenario in which the system allocates and schedules incoming tasks according to existing computing resources in the process of constructing the state channel in the blockchain environment. The scenario consists of 10 processing units and 100–1000 federated learning tasks with different requirements on service quality. Each unit has a computation rate of 500–2000 MIPS and an overhead of 5–20 per unit of time. The workload length of each task is 100–4000 MI,

the budget is 10–50, and the maximum completion time of the task that can be tolerated is 5–10 s. The setting of specific experimental parameters is shown in Table 2.

The Actor–Critic algorithm has 100 epochs, and 5000 updates of network parameters are performed in each epoch. The buffer size is set to 2000, and the learning rate of both the Actor network and Critic network is 1×10^{-4}. The specific algorithm parameter settings are shown in Table 3.

In this paper, several algorithms are selected to compare with the proposed algorithm (A2C). (1) In the random selection algorithm, each federated learning task is randomly allocated a block of computing resources. (2) The greedy algorithm only considers scheduling schemes that can minimize the maximum completion time of the system. (3) Genetic algorithm (GA) [21] is designed and proposed according to the biological evolution law in nature and simulates the natural selection and biological evolution process, which is a common method to solve the problem of objective optimization.

Table 2. Experimental Environment Parameters.

Parameters	Settings
NUMBER OF COMPUTING UNITS	10
COMPUTATION RATE	500–2000 MIPS
COST WITHIN A UNIT OF TIME	5–20
NUMBER OF TASKS	100–1000
TASK LENGTH	100–4000 MI
TASK BUDGET	10–50
MAXIMUM COMPLETION TIME OF TASK	5–10 s

Table 3. Parameters of A2C Algorithm.

Parameters	Settings
Epoch (M)	100
Step per epoch (T)	50,000
Buffer size (S)	2000
Batch size (N)	64
γ	0.9
k	100
α	1–e4
β	1–e4
τ	0.5
ε	1

6.2. Analysis of Results

In this paper, several evaluation indicators of the algorithm are set as follows:

- $maxMakespan$, the maximum completion time of the system, is used to evaluate the efficiency of the system, as defined in Equation (5).
- $avgCost$, the average task cost, which evaluates the cost of a task, is defined as follows:

$$avgCost = \frac{1}{N_{task}} \sum_{i=1}^{N_{task}} \sum_{j=1}^{N_{pu}} ECT_i * COST_j * x_{i,j} \tag{22}$$

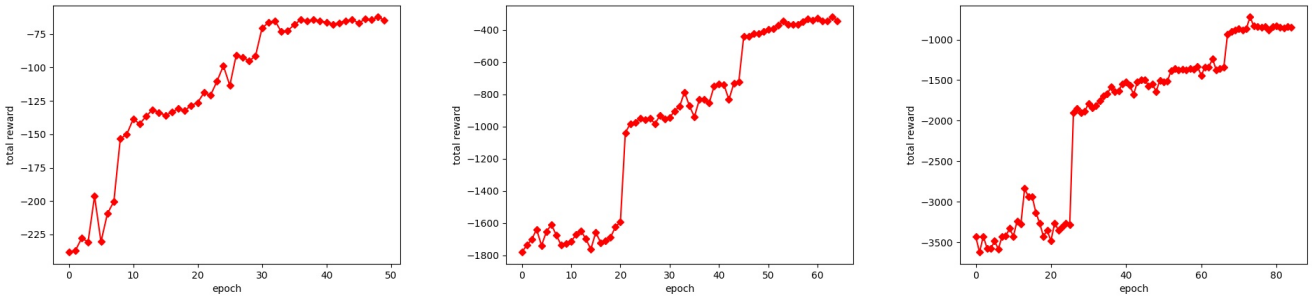
- $avgCompletedTime$, the average task competition time, is defined as follow:

$$avgCompletedTime = \frac{1}{N_{task}} * \sum_{i=1}^{N_{task}} \frac{WorkLoad_i}{\sum_{j=1}^{Npu} x_{i,j} * E_j} \tag{23}$$

- $mSLA$, the percentage of the tasks that meet the requirements on service quality in the total tasks, evaluates the ability of the algorithm to meet the requirements of tasks on service quality, which is defined as follows:

$$mSLA = \frac{N_{mSLA}}{N_{task}} \tag{24}$$

where N_{mSLA} represents the proportion of tasks that meet the requirements on service quality among all tasks. This paper assumes that the indicators of the requirements of tasks on service quality are $TCost$ and ECT; that is, the tasks that meet the two indicators are defined as the tasks that meet the quality of service requirements.



(**a**) data sets of 100 tasks (**b**) data sets of 100 tasks (**c**) data sets of 100 tasks

Figure 4. The total reward of the A2C on data sets of different numbers of tasks.

Figure 4 shows the change of the total reward of the Actor–Critic algorithm on 100, 500, and 1000 test tasks set, respectively. As can be seen from the figure, the total reward of the model obtained at the initial stage of training is low on the task set, which indicates that the model needs to be trained a certain number of times to obtain a better resource allocation scheme. In addition, it can be found that the convergence rate of the algorithm on the different number of tasks is not the same. In general, the more tasks the algorithm converges, the slower the algorithm converges. When the number of training iterations reaches 40, 50, and 70, the total reward of the algorithm on 100, 500, and 1000 test tasks set gradually becomes stable.

Table 4. The performance of the four scheduling algorithms on each metric. The columns give the results of the algorithms on the different evaluation metrics and the rows indicate the different scheduling algorithms. Each metric is further divided into results on sets of 100, 500, and 1000 number of tasks.

Scheduling	**maxMakespan**			**avgCost**			**avgCompletedTime**			**mSLA**		
Algorithms	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**100**	**100**	**500**	**1000**
Random	164	833	1537	7.58	7.34	6.95	18.71	19.05	18.85	0.48	0.45	0.44
Greed	27	103	224	4.14	4.02	4.88	16.3	16.79	16.83	0.51	0.54	0.55
GA	62	374.2	722	2.3	3.25	4.01	8.6	10.54	11.93	0.86	0.84	0.82
A2C	47	237	503	1.05	1.83	2.57	4.27	5.28	6.2	0.96	0.93	0.91

This paper illustrates the advantages of DRL in resource scheduling by comparing it with other three heuristic and meta-heuristic resource-scheduling algorithms. Table 4 shows the performance of four different resource-scheduling algorithms on the different number of task sets. The indexes are $maxMakespan$ (the maximum completion time of the system), $avgCost$ (the average task cost, which evaluates the cost of a task), $avgCompletedTime$ (the average task competition time), and $mSLA$ (the proportion of tasks that meet the requirements on service quality among all tasks). The four algorithms are the random selection algorithm, greedy algorithm, GA (genetic algorithm), and A2C (the proposed algorithm based on deep reinforcement learning).

As for the performance of the *maxMakespan* index, it can be seen from the table that the maximum completion time of the system obtained by using a random scheduling algorithm on 100, 500, and 1000 task sets is the largest among the four scheduling algorithms and is proportional to the number of tasks. The GA algorithm and A2C algorithm come second, and the greedy strategy has the least. This trend also increases with the number of tasks, and the gap between different algorithms becomes more and more obvious. Figure 5 shows the histogram of different scheduling algorithms on *maxMakespan*, which can intuitively show the advantages and disadvantages of different algorithms on *maxMakespan*.

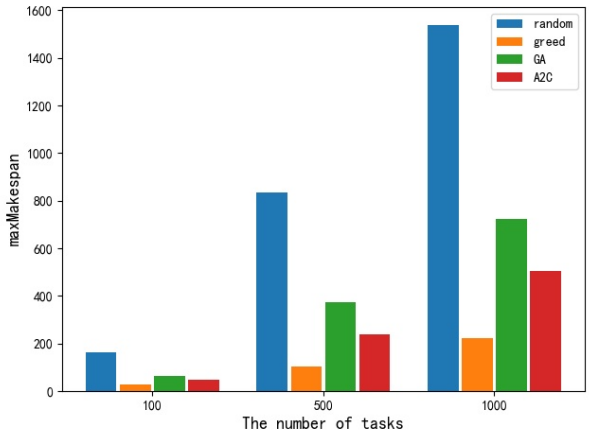


Figure 5. Comparison of different algorithms in *maxMakespan*.

For *avgCost* and *avgCompletedTime*, different from *maxMakespan*, the greedy algorithm performed poorly in these two indexes. The *avgCost* obtained by the greedy algorithm is only slightly better than that of the random selection scheduling algorithm, but the gap is not big. The performance of the other three algorithms on this index from bad to good is the random selection algorithm, GA algorithm, and A2C algorithm, respectively. Figures 6 and 7, respectively, show the histogram of performance of different scheduling algorithms at *avgCost* and *avgCompletedTime*.

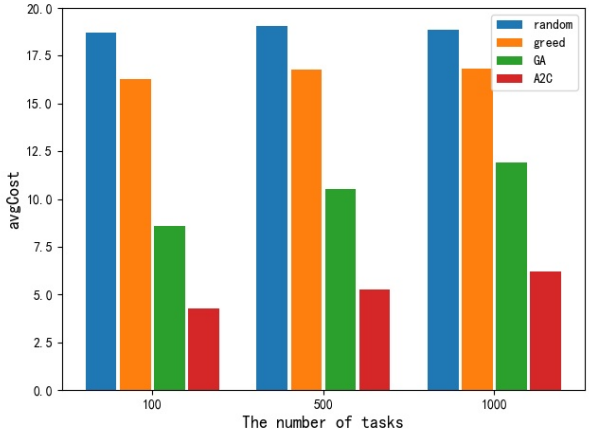


Figure 6. Comparison of different algorithms in avgCost.

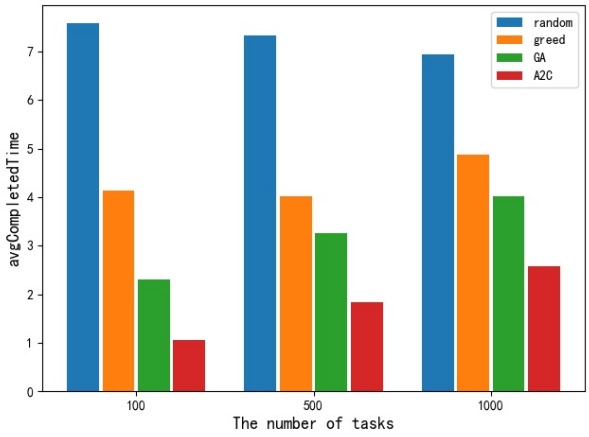


Figure 7. Comparison of different algorithms in *avgCompletedTime*.

As for the *mSLA* index, it can be seen that the random selecting scheduling algorithm on the test set can only ensure that less than 50% of the tasks meet the requirements of service quality. The greedy algorithm and random selection algorithm are not much different. The GA algorithm can ensure that about 80% of the tasks meet the quality of service requirements. The A2C algorithm can ensure that more than 90% of the tasks can meet the requirements under different task numbers, but the proportion decreases with the increase of tasks. Figure 8 shows the histogram of the performance of different resource-scheduling algorithms on *mSLA*.

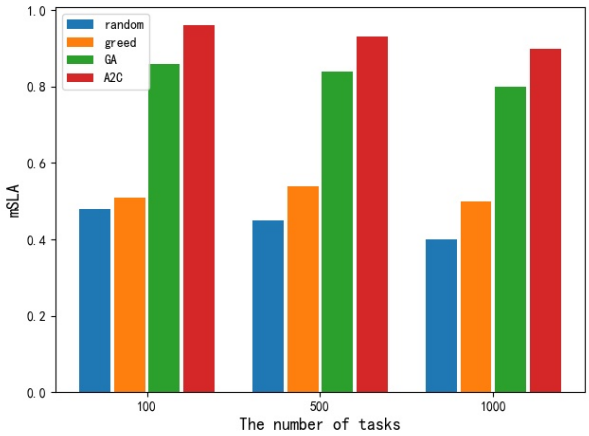


Figure 8. Comparison of different algorithms in *mSLA*.

Combined with the pictures and the table, random selection scheduling has the worst effect on any index because of its randomness and uncertainty. Although the greedy algorithm has an excellent performance in the completion time of the system, it can not well meet the service quality requirements of the task, and the effect of other evaluation indicators is not good. This is because the greedy algorithm only considers the maximum completion time of the system and does not take into account other factors such as the task's quality of service requirements. The A2C algorithm and GA algorithm not only consider the maximum completion time of the system but also consider the task quality of service requirements and other constraints. The evaluation on the different number of tasks and different indexes shows that the A2C algorithm has a better effect than the traditional GA algorithm on this problem model. For different system sizes and complex environments, the A2C algorithm based on deep reinforcement learning can constantly optimize its scheduling strategy through the feedback of the environment, and after a certain degree of pre-training, it can conduct efficient and fast scheduling. Experiments

show that the proposed algorithm can reduce the maximum completion time of the system based on satisfying the requirements of task quality of service, which makes the whole blockchain system more efficient in scheduling federated learning and training tasks.

7. Conclusions

This paper designs a federated learning multi-task scheduling mechanism based on a trusted computing sandbox. First of all, a system framework for completing federated learning tasks in the blockchain environment is proposed. The blockchain reasonably allocates resources for each federated learning task and constructs a computing sandbox as a state channel. The federated learning model training process is carried out in the channel and supervised by all nodes. Secondly, considering the factors such as the resource heterogeneity of each participating node, system efficiency, and the requirements of tasks on service quality, an optimization problem model of resource allocation and scheduling in the blockchain scenario was constructed. Finally, the problem model was constructed as an MDP model, and a resource-scheduling algorithm based on Actor–Critic was designed to solve the problem. The experimental results show that the proposed algorithm has good convergence under different task number scenarios, and it can reduce the maximum completion time of the system and improve the efficiency of the system while meeting the requirements of tasks on service quality.

For future work, the privacy and security issues behind the value sharing of a large amount of data will be paid more and more attention. Federated learning and blockchain technology will play an increasingly important role in the field of data privacy and security through the advantages of local data and the efficient and safe data-sharing method of decentralization. How to protect user data privacy through federated learning in the decentralized mode will become the focus of future research. In the next step, we will study how to realize the discovery and supervision of malicious behaviors of nodes in the computing sandbox during the training of federated learning to prevent privacy leakage and attacks in time.

Author Contributions: Conceptualization, H.L.; Methodology, H.L.; Software, H.L. and H.Z.; Validation, H.L. and H.Z.; Formal analysis, H.L. and H.C.; Investigation, H.L. and H.Z.; Resources, H.Z.; Data curation, H.C.; Writing—original draft, H.Z.; Writing—review and editing, Y.Y., A.X., S.Y. and J.C.; Visualization, H.C., J.H., S.Y., J.C. and S.G.; Supervision, H.C., Y.Y., J.H., A.X., S.Y., J.C. and S.Y.; Project administration, Y.Y., J.H., A.X., S.Y., J.C. and S.G.; Funding acquisition, Y.Y., J.H., A.X. and S.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by State Grid Corporation of China Science and Technology Project "Research and application of industry chain finance key technology based on blockchain" (5211DS21NOOU).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nakamoto, S.. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Bus. Rev.* **2008**, 21260.
2. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. *Artif. Intell. Stat.* **2017**, 1273–1282.
3. Praneeth, K.S.; Kale, S.; Mohri, M.; Reddi, S.J.; Stich, S.U.; Theertha, S.A. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. *arXiv* **2019**, arXiv-1910 .
4. Li, T.; Sahu, A.K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; Smith, V. Federated optimization in heterogeneous networks. *Proc. Mach. Learn. Syst.* **2020**, 429–450.
5. Acar, D.A.; Zhao, Y.; Navarro, R.M.; Mattina, M.; Whatmough, P.N.; Saligrama, V. Federated learning based on dynamic regularization. *arXiv* **2021**, arXiv:2111.04263.
6. He C; Tan C; Tang H; Qiu S; Liu J. Central server free federated learning over single-sided trust social networks. *arXiv* **2019**, arXiv:1910.04956.
7. Vanhaesebrouck, P.; Bellet, A.; Tommasi, M. Decentralized collaborative learning of personalized models over networks. *Artif. Intell. Stat.* **2017**, 509–517.

8. Bellet, A.; Guerraoui, R.; Taziki, M.; Tommasi, M. Personalized and private peer-to-peer machine learning. *Int. Conf. Artif. Intell. Stat.* **2018**, 473–481.
9. Roy, A.G.; Siddiqui, S.; Pölsterl, S.; Navab, N.; Wachinger, C. Braintorrent: A peer-to-peer environment for decentralized federated learning. *arXiv* **2019**, arXiv:1905.06731.
10. Hu, C.; Jiang, J.; Wang, Z. Decentralized federated learning: A segmented gossip approach. *arXiv* **2019**, arXiv:1908.07782.
11. Li, Y.; Chen, C.; Liu, N.; Huang, H.; Zheng, Z.; Yan, Q. A blockchain-based decentralized federated learning framework with committee consensus. *IEEE Netw.* **2020**, *35*, 234–241. [CrossRef]
12. Pappas, C.; Chatzopoulos, D.; Lalis, S.; Vavalis, M. IPLS: A framework for decentralized federated learning. In Proceedings of the 2021 IFIP Networking Conference, Espoo, Finland, 21–24 June 2021; pp. 1–6
13. Li, Y.; Yu, M.; Li, S.; Avestimehr, S.; Kim, N.S.; Schwing, A. Pipe-SGD: A decentralized pipelined SGD framework for distributed deep net training. *Adv. Neural Inf. Process. Syst.* **2018**, *31* .
14. Zheng, Z.; Xie, S.; Dai, H.N.; Chen, X.; Wang, H. Blockchain challenges and opportunities: A survey. *Int. J. Web Grid Serv.* **2018**, *14*, 352–375. [CrossRef]
15. Podgorelec, B.; Heričko, M.; Turkanović, M. State channel as a service based on a distributed and decentralized web. *IEEE Access* **2020**, *8*, 64678–64691. [CrossRef]
16. Qu, Y.; Gao, L.; Luan, T.H.; Xiang, Y.; Yu, S.; Li, B.; Zheng, G. Decentralized privacy using blockchain-enabled federated learning in fog computing. *IEEE Internet Things J.* **2020**, *7*, 5171–5183. [CrossRef]
17. Zhao, Y.; Zhao, J.; Jiang, L.; Tan, R.; Niyato, D.; Li, Z.; Lyu, L.; Liu, Y. Privacy-preserving blockchain-based federated learning for IoT devices. *IEEE Internet Things J.* **2020**, *8*, 1817–1829. [CrossRef]
18. Yahyaoui, H.; Maamar, Z.; Lim, E.; Thiran, P. Towards a community-based, social network-driven framework for Web services management. *Future Gener. Comput. Syst.* **2013**, *29*, 1363–1377. [CrossRef]
19. Pradhan, P.; Behera, P.K.; Ray, B.N. Modified round robin algorithm for resource allocation in cloud computing. *Procedia Comput. Sci.* **2016**, *85*, 878–890. [CrossRef]
20. Maheswaran, M.; Ali, S.; Siegel, H.J.; Hensgen, D.; Freund, R.F. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *J. Parallel Distrib. Comput.* **1999**, *59*, 107–131. [CrossRef]
21. Yu, J.; Buyya, R. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. *Workshop Work. Support -Large-Scale Sci.* **2006**, 1–10 .
22. Li, H.; Zhu, G.; Zhao, Y.; Dai, Y.; Tian, W. Energy-efficient and QoS-aware model based resource consolidation in cloud data centers. *Clust. Comput.* **2017**, *20*, 2793–2803. [CrossRef]
23. Abualigah, L.; Diabat, A. A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments. *Clust. Comput.* **2021**, *24*, 205–223. [CrossRef]
24. Strumberger, I.; Tuba, M.; Bacanin, N.; Tuba, E. Cloudlet scheduling by hybridized monarch butterfly optimization algorithm. *J. Sens. Actuator Netw.* **2019**, *8*, 44. [CrossRef]
25. Prem, J.T.; Pradeep, K. A multi-objective optimal task scheduling in cloud environment using cuckoo particle swarm optimization. *Wirel. Pers. Commun.* **2019**, *109*, 315–331. [CrossRef]
26. Guo, W.; Tian, W.; Ye, Y.; Xu, L.; Wu, K. Cloud resource scheduling with deep reinforcement learning and imitation learning. *IEEE Internet Things J.* **2020**, *8*, 3576–3586. [CrossRef]
27. Tuli, S.; Ilager, S.; Ramamohanarao, K.; Buyya, R. Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks. *IEEE Trans. Mob. Comput.* **2020**, *21*, 940–954. [CrossRef]
28. Feng, J.; Yu, F.R.; Pei, Q.; Chu, X.; Du, J.; Zhu, L. Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach. *IEEE Internet Things J.* **2019**, *7*, 6214–6228. [CrossRef]
29. Dong, T.; Xue, F.; Xiao, C.; Li, J. Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5654. [CrossRef]
30. 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.* **2020**, *7*, 9399–9412. [CrossRef]
31. Kardani-Moghaddam, S.; Buyya, R.; Ramamohanarao, K. Adrl: A hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 514–526. [CrossRef]
32. Ding, D.; Fan, X.; Zhao, Y.; Kang, K.; Yin, Q.; Zeng, J. Q-learning based dynamic task scheduling for energy-efficient cloud computing. *Future Gener. Comput. Syst.* **2020**, *108*, 361–371. [CrossRef]
33. Sun, G.; Zhan, T.; Owusu, B.G.; Daniel, A.M.; Liu, G.; Jiang, W. Revised reinforcement learning based on anchor graph hashing for autonomous cell activation in cloud-RANs. *Future Gener. Comput. Syst.* **2020**, *104*, 60–73. [CrossRef]
34. Weng, J.; Chen, H.; Yan, D.; You, K.; Duburcq, A.; Zhang, M.; Su, Y.; Su, H.; Zhu, J. Tianshou: A highly modularized deep reinforcement learning library. *arXiv* **2021**, arXiv:2107.14171.