Blockchain-based Edge Resource Sharing for Metaverse

Zhilin Wang†, Qin Hu† (Corresponding author), Minghui Xu§, Honglu Jiang†
†Department of Computer & Information Science, Indiana University-Purdue University Indianapolis, USA
§School of Computer Science & Technology, Shandong University, China
Email: {wangzhil, qinh}@iu.edu, mhxu@sdu.edu.cn, jiangh34@miamioh.edu

Abstract—Although Metaverse has recently been widely studied, its practical application still faces many challenges. One of the severe challenges is the lack of sufficient resources for computing and communication on local devices, resulting in the inability to access the Metaverse services. To address this issue, this paper proposes a practical blockchain-based mobile edge computing (MEC) platform for resource sharing and optimal utilization to complete the requested offloading tasks, given the heterogeneity of servers’ available resources and that of users’ task requests. To be specific, we first elaborate the design of our proposed system and then dive into the task allocation mechanism to assign offloading tasks to proper servers. To solve the multiple task allocation (MTA) problem in polynomial time, we devise a learning-based algorithm. Since the objective function and constraints of MTA are significantly affected by the servers uploading the tasks, we reformulate it as a reinforcement learning problem and calculate the rewards for each state and action considering the influences of servers. Finally, numerous experiments are conducted to demonstrate the effectiveness and efficiency of our proposed system and algorithms.

Index Terms—Metaverse, mobile edge computing, blockchain, reinforcement learning

I. INTRODUCTION

Metaverse, which is considered as the next generation of the Internet, has attracted researchers’ attention recently [1], [2]. In Metaverse, people can interact with the virtual world through technologies like virtual reality and augmented reality. Currently, people typically access the servers of Metaverse providers through Metaverse local devices, such as head-mounted glasses, to get the Metaverse services. These devices are required to assist users in accessing and experiencing Metaverse services, as well as to process resource-intensive computing tasks locally. However, using Metaverse local devices for computing faces severe challenges: 1) the computing and communication resources of the devices are limited; 2) the locations of devices performing local computing are dispersed and the devices may be constantly moving.

Fortunately, there is one existing solution for addressing these challenges, namely mobile edge computing (MEC) [3], [4]. Specifically, local devices can offload their computing tasks to proxy MEC servers, e.g., base stations; then, the MEC servers finish those tasks and return the results to local devices. Since the MEC servers are located close to local devices and usually have enough computing resources, their involvement can reduce the latency of communication and computing, thus providing low-latency and high-quality Metaverse services.

In practice, the MEC servers in Metaverse are usually responsible for computing multiple tasks from different users. Since their resources are not infinite, one single MEC server may not be able to handle those offloading tasks in time, leading to low quality of Metaverse services. One possible solution is to involve other MEC servers with extra resources to work together for offloading computing in Metaverse, which makes it necessary to establish a secure resource trading platform for edge servers. To that aim, we propose a blockchain system running on MEC servers to form a distributed computing framework, named blockchain-based MEC platform, which enables resource integration and optimal utilization among MEC servers in a trustless environment.

Currently, there is no existing study focusing on implementing blockchain-based MEC for resource sharing and optimization in Metaverse. Although there exist several studies about MEC in Metaverse, they focus on the latency analysis [5] and incentive mechanism design [6]. While for research about blockchain-based MEC [7], [8], no one has considered solving the resource sharing problem in MEC.

To fill this gap, the proposed blockchain-based MEC platform aims to assist resource sharing and optimization in Metaverse via trading offloading tasks in a transparent but secure way, so that more offloading tasks from Metaverse users can be finished timely. This platform comprises multiple Metaverse users, MEC servers, and a consortium blockchain system running the practical Byzantine fault tolerance (PBFT) consensus protocol [9], [10]. And there are four main procedures in the proposed system, i.e., data submission, task allocation, offloading computing, and payment of tasks. As the pivotal step, task allocation faces a critical challenge brought by the heterogeneity of multi-task requests from users. Specifically, given the limited computing and communication resources of each MEC server and offloading tasks with different price policies, data sizes, and completion time requirements, our proposed system needs to assign the requested multiple tasks to multiple servers under various constraints. In addition, since the offloading tasks are time-sensitive, the task allocation step is expected to make the (near) optimal decisions as fast as possible so as to reduce the latency of the whole system.

To address these challenges, we design a learning-based task allocation mechanism. Specifically, we first formulate the task
allocation issue as an integer-programming problem; then, we analyze the utilities of different decisions by considering the resource constraints of servers and the time requirements of tasks; according to our analysis, the multiple tasks allocation (MTA) problem is NP-complete, and the time complexity is too high, so we design a learning-based algorithm to find its approximate optimal value in polynomial time. Since the objective function and constraints of the MTA problem are conditional, which are affected by the servers uploading the tasks to blockchain, in our proposed learning-based solution, we first transform the MTA problem into a reinforcement learning problem and then calculate the rewards for each state and action considering the influences of the server source of tasks.

To the best of our knowledge, we are the first to implement blockchain-based MEC in Metaverse. The main contributions of this paper are summarized as below:

- We are the first to propose a practical blockchain-based MEC platform for resource sharing and optimal utilization in Metaverse. Our proposed system can satisfy the demands of low latency, multiple requests, energy efficiency, incentive compatibility, and data privacy protection for Metaverse users.
- We design a task allocation mechanism for edge resource sharing. Under the heterogeneous constraints, we first formulate an MTA problem for reasonably distributing offloading tasks among multiple MEC servers.
- We propose a learning-based solution to find the approximate optimal solution for the MTA problem with polynomial time complexity. We speed up the learning process via identifying available actions for each state.
- We conduct extensive experiments to verify the effectiveness, efficiency, and validity of our proposed system, mechanisms, and algorithms.

II. SYSTEM MODEL

A. System Overview

Our proposed blockchain-based edge resource sharing platform is illustrated in Fig. 1 for supporting the Metaverse applications, consisting of mobile devices as Metaverse users, MEC servers, and the consortium blockchain running with practical Byzantine fault tolerance (PBFT) consensus. Here we assume that users in our considered system are devices with offloading requests to MEC servers. Based on the arrival time of offloading tasks in the blockchain network, we assign a specific task number to each of them. Specifically, we define \( T = \{ t_1, \cdots, t_j, \cdots, t_m \} \) as the set of offloading tasks from users with \( m \) denoting the number of all offloading tasks. Let \( S = \{ s_1, \cdots, s_i, \cdots, s_n \} \) denote the set of MEC servers in the Metaverse with \( n \) being the total number of servers.

The workflow of our proposed system can be described as below: 1) Data Submission: The Metaverse users upload their raw data and the descriptions of tasks to their nearest MEC servers. We use \( R_j < p_j, D_j, \tau_{e,j} > \) to denote the description of task \( t_j \), where \( p_j \) is the unit price of per CPU cycle of computing for finishing \( t_j \), \( D_j \) is the data size of \( t_j \), and \( \tau_{e,j} \) is the time requirement for finishing \( t_j \). 2) Task Allocation: Once server \( s_i \) receives the data and \( R_j \) from users, the data is stored locally and \( R_j \) is submitted to the blockchain for task allocation which will be elaborated in the following sections. 3) Offloading Computing: According to the task allocation results, server \( s_i \) will either send the data of tasks it received to appropriate servers or finish the computing of the tasks locally. Then, the servers start to work on the allocated tasks and will broadcast the computing results to the blockchain network when the tasks are completed. 4) Payment of Tasks: The users first pay offloading computing fees through the blockchain network where they can also get the computing results later. Each server can get its own payment through the coordination of the blockchain.

To protect the privacy of the computing results, we can employ the asymmetric encryption technique, such as Rivest–Shamir–Adleman (RSA) algorithm [11], in this system. Specifically, at the beginning of each round, the users submit the public keys generated by RSA along with the offloading task requests while keep their private keys secret. After the tasks are finished by the MEC servers, the computing results will be encrypted with the public keys of the corresponding users. Then, the users can download the encrypted results via accessing the blockchain and decrypt the results with their private keys. Since the amount of tasks received by each server can be different, we define the tasks submitted by server \( s_i \) as \( T_{s,i} = \{ t_1, t_2, \cdots, t_{m'} \} \), where \( m' \leq m \) is the number of tasks from \( s_i \). For example, if \( t_1, t_3, \) and \( t_20 \) are submitted by \( s_1 \), then we have \( T_{s,1} = \{ t_1, t_3, t_{20} \} \).

B. Consortium Blockchain

The consortium blockchain is running on the MEC servers, where all servers are authorized nodes and thus can be trusted. Besides, as mentioned earlier, we implement PBFT, a lightweight consensus protocol in blockchain, to assist servers in reaching consensus for generating blocks to help disseminate task descriptions and allocation results, as well as enforce payment distribution in a round-by-round manner.

Here is the workflow of the blockchain network in our system: 1) At the beginning of each round, each server publishes its available resources for finishing tasks, including
both computing and communication resources (see Section III for details), and uploads received task descriptions $R_j$ to the blockchain network. 2) All of the above-mentioned information will be broadcast in the blockchain network so that the leader node can allocate the received tasks to appropriate servers with the help of the task allocation mechanism detailed in the next section. Then the allocation decisions, resource information of the MEC servers, and task descriptions will be packaged into a new block. 3) If $s_i$ is assigned to process the tasks submitted by itself, it can start processing right away; otherwise, it has to transfer the received data to other servers for finishing the offloading tasks. 4) After tasks are finished, the results will be broadcast so as to be recorded on the blockchain, and the servers will be paid accordingly.

Overall, the blockchain offers a decentralized and trusted platform to conduct resource sharing for MEC empowered Metaverse applications, thus enabling offloading computing for Metaverse users to overcome the challenges posed by the Metaverse applications, thus enabling offloading computing to conduct resource sharing for MEC empowered blockchain, and the servers will be paid accordingly.

C. Utility Model

After $t_j$ is completed, server $s_i$ will receive the payment from users, denoted by $p_j \mu_{i,j}$ for task $t_j$, which is the product of unit price per CPU cycle and the total number of consumed CPU cycles. Then, the utility of $s_i$ regarding finishing $t_j$ in the system is the difference between the received payment and the local cost.

Since there are two possible task allocation results for any server $s_i$ with respect to processing task $t_j$, i.e., computing $t_j$ locally or not, which will significantly affect the utility of $s_i$, we define an indicator $I_{i,j}$ to capture this: if $t_j$ is assigned to $s_i$, then $I_{i,j} = 1$; otherwise, $I_{i,j} = 0$. If $t_j \in T_{s,i}$, the utility can be expressed as $U_{i,j} = (p_j \mu_{i,j} - E_{comp}^i) I_{i,j} + \lambda p_j \mu_{i,j} - E_{comm}^i (1 - I_{i,j})$, where $\lambda$ is the intermediary fee paid by the server finishing $t_j$, with $\lambda$ being a predefined constant parameter that can be known globally; and the total time consumption is $T_i^j = T_{i,j}^{comp} I_{i,j} + T_{i,j}^{comm} (1 - I_{i,j})$. While if $t_j \notin T_{s,i}$, the definition is $U_{i,j}^N = [(1 - \lambda) p_j \mu_{i,j} - E_{comp}^i] I_{i,j}$, and the total time consumption is $T_i^j = T_{i,j}^{comp} I_{i,j} + T_{i,j}^{comm}$, where $T_{i,j}^{comm}$ is the transmission time of server submitting $t_j$. Thus, we can get the utility function and time consumption as: if $t_j \in T_{s,i}$, $U_{i,j} = U_{i,j}^I$ and $T_{i,j} = T_{i,j}^I$; otherwise, $U_{i,j} = U_{i,j}^N$ and $T_{i,j} = T_{i,j}^N$.

D. Problem Formulation

Recall the goal of task allocation for achieving resource integration and optimal utilization in our system, we would like to make sure that more tasks can be proceed in time and thus servers can get more rewards. So we can formulate it into a multi-task allocation (MTA) problem as follows:

$$\text{MTA} : \arg \max_{\mathbf{x}_{i,j}} \sum_{i=1}^{n} \sum_{j=1}^{m} U_{i,j} \quad \text{s.t.} \quad C_1 : \sum_{j=1}^{m} \mu_{i,j} I_{i,j} \leq \mu_i, \quad C_2 : T_{i,j} \leq \tau_{c,j}, \quad C_3 : \sum_{j=1}^{m} I_{i,j} \leq 1, \quad C_4 : i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\}, 0 < n \leq m.$$
Theorem III.1. MTA is an NP-complete problem.

Proof. The MTA problem can be described as a complex multiple knapsacks problem (MKP), which has been proved to be NP-complete in [13]. Specifically, MKP is defined as: let \( T' = \{ t_1, t_2, \ldots, t_m' \} \) denote the set of \( m \) items, and let \( S' = \{ s_1, s_2, \ldots, s_n' \} \) as the set of \( n \) knapsacks, and \( m' > n' \). Each item \( t_i' \) has its weight \( D_i' \) and value \( p_i' \) and each knapsack has its maximum weight \( D_i'. \) The goal is to decide where should each item be placed in the knapsacks so that the total utility is maximized. In other words, it aims to solve the following problem: \( \max_{D, p, i,j} \sum_{i,j} D_i' p_i' \) \( i,j \) is an indicator: when \( i,j = 1 \), item \( t_i' \) will be assigned to knapsack \( s_i' \), and otherwise, \( i,j = 0 \). The time complexity is \( O(n^m) \), which means MKP cannot be solved in polynomial time. MKP can be reduced to the simplified MTA problem. In the original MTA problem, we need to allocate multiple tasks to multiple servers with computing resource and communication constraints. The objective of MTA is to maximize the total utility, i.e., \( \max_{D, p, i,j} \sum_{i,j} U_{i,j} \), where \( U_{i,j} \) is the utility. If we remove the constraints that every server offers the same computing and communication resources for each task, we get the simplified MTA problem which is equivalent to MKP. As we can see, the simplified MTA also has the time complexity of \( O(n^m) \), so MTA is an NP-complete problem. \( \square \)

IV. LEARNING-BASED SOLUTION FOR MTA

Based on the analysis in Section III-D, we know that the MTA problem is NP-complete, so we need to design a computationally efficient algorithm to solve it. In this section, we design a learning-based algorithm with \( \epsilon \)-greedy strategy.

A. Problem Reformulation

To begin with, we need to reformulate the MTA problem based on the Q-learning algorithm [14], [15]. There are three main components in Q-learning, i.e., state space, action space, and reward function, which are detailed as follows.

1) State Space: We denote the state space of the agent, i.e., the MEC server executing the Q-learning algorithm to allocate tasks, as \( \Omega < T, P, D, T_a > \). Specifically, \( T = \{ t_1, t_2, \ldots, t_m \} \) is the set of tasks, \( P \) is the set of the unit prices of tasks, i.e., \( P = \{ p_1, p_2, \ldots, p_m \} \), \( D = \{ D_1, D_2, \ldots, D_m \} \) is the set of data sizes of tasks, and the set of execution time is defined as \( T_e = \{ t_{e,1}, t_{e,2}, \ldots, t_{e,m} \} \). In other words, the state space is composed of all tasks with the corresponding descriptions, including prices, data sizes, and execution time requirements. Thus, \( \Omega \) is a matrix with \( m \) rows and 4 columns. The agent selects an action for each state based on the current task requirements and resource conditions of all servers. Once the action is chosen at a state, the agent will turn to the next state to conduct action selection.

2) Action Space: Since servers have different amount of available resources, such as total CPU cycles, CPU cycle frequencies, and communication bandwidth, we can denote the action space as \( A < S, M, F, B, H, G, \alpha, \Theta > \). In detail, \( S \) is the set of servers, and \( M = \{ \mu_1, \mu_2, \ldots, \mu_n \} \) is the set of total available CPU cycles, and \( F = \{ f_1, f_2, \ldots, f_n \} \) is the set of CPU frequencies, and \( B = \{ B_1, B_2, \ldots, B_n \} \) is the set of communication bandwidth; as for \( H = \{ H_1, H_2, \ldots, H_n \} \) and \( G = \{ G_1, G_2, \ldots, G_n \} \), they are the sets of transmission power and channel power gain, respectively; and \( \alpha = \{ \alpha_1, \alpha_2, \ldots, \alpha_n \} \) is the set of the parameter correlated to the CPU architectures and \( \Theta = \{ \theta_1, \theta_2, \ldots, \theta_n \} \) is the set of CPU cycles required for processing one data sample. Specifically, there are two statuses for each action, i.e., selected and not selected, and the agent can choose only one action in one state while one action can be selected multiple times in all the states. This is to ensure that one task can only be assigned to one server; however, one server could process multiple tasks if it has sufficient resources. In this way, we can know that the action space is an \( n \times 8 \) matrix.

3) Reward Function: The objective of MTA problem is to maximize the utility by allocating tasks to proper servers, so the rewards here are defined by the utilities. In other words, the rewards are determined by the allocation decision, resource conditions, and time constraints. Besides, the rewards would also be affected by where the task is submitted from to the blockchain network. Thus, in the design of the reward function, we need to consider all of these aspects.

We evaluate whether our system can process \( t_j \) by two criteria: 1) server \( s_i \) has enough computing power to be devoted to the computation, i.e., \( \mu_i \leq \mu_t \); and 2) \( s_i \) is able to process the task \( t_j \) within the required time constraint, i.e., \( T_{i,j} \leq \tau_{e,j} \). The second criteria is C2 while the first criteria is a weak C1. And C1 can only be used when selecting actions and updating Q-value, which will be discussed in Section IV-B.

If \( t_j \in T_{s_i} \), the reward function can be expressed as:

\[
U_{i,j} = \begin{cases} 
 p_j \mu_{i,j} - E_{i,j}^{\text{comp}}, & \mu_{i,j} \leq \mu_t \text{ and } T_{i,j} \leq \tau_{e,j}, \\
\lambda p_j \mu_{i,j} - E_{i,j}^{\text{comp}}, & \text{otherwise.}
\end{cases}
\]

When \( t_j \) is from \( s_i \), if \( s_i \) is not assigned to process \( t_j \), it has to transmit \( t_j \) to another server and obtain some intermediary fee, so the reward is \( \lambda p_j \mu_{i,j} - E_{i,j}^{\text{comp}} \), but if \( s_i \) is able to process \( t_j \), it can get the payment \( p_j \mu_{i,j} \) at the cost of the computing energy consumption \( E_{i,j}^{\text{comp}} \), and thus, the reward is \( U_{i,j} = p_j \mu_{i,j} - E_{i,j}^{\text{comp}} \).

If \( t_j \notin T_{s_i} \), the reward function is as below:

\[
U_{i,j} = \begin{cases} 
(1-\lambda)p_j \mu_{i,j} - E_{i,j}^{\text{comp}}, & \mu_{i,j} \leq \mu_t \text{ and } T_{i,j} \leq \tau_{e,j}, \\
0, & \text{otherwise.}
\end{cases}
\]

The logic of (2) is similar to (1): when \( t_j \) is not from \( s_i \), if \( s_i \) has the capability to process \( t_j \), it can get the reward the same as in (1) but has to pay the intermediary fee; and it will get nothing if it cannot process this task, which means that \( A_{i,j} \) as one action cannot be chosen in state \( H_{i,j} \).

Based on the above analysis, we know that the reward functions are the transformation of the objective function under certain constraints. For simplicity, we summarize the calculation of reward functions in Algorithm 1. To get an \( n \times m \) matrix \( U = \{ U_{1,1}, U_{1,2}, \ldots, U_{1,m}, \ldots, U_{n,m} \} \) containing the reward value for each state and each action, we need to calculate the energy consumption and time cost for both computing and communication processes (Lines 3-6). And
then we can calculate and return the rewards based on (1) and (2) (Lines 7-25).

Algorithm 1 StateActionReward

Require: $\Omega$, $A$

Ensure: $U$

1: for $i \in \{1, \ldots, n\}$ do
2: for $j \in \{1, \ldots, m\}$ do
3: $E_{i,j}^{\text{comp}} \leftarrow \alpha_i \mu_{i,j} f_i$
4: $T_{i,j}^{\text{comp}} \leftarrow \mu_{i,j}$
5: $T_{i,j}^{\text{comm}} \leftarrow \hat{B}_j$
6: $E_{i,j}^{\text{comm}} \leftarrow H(T_{i,j}^{\text{comm}})$
7: if $s_j \in T_{i,j}$ then
8: $T_{i,j} \leftarrow T_{i,j}^{\text{comp}} + T_{i,j}^{\text{comm}} (1 - \mathbbm{1}_{i,j})$
9: if $(\mu_{i,j} \leq \mu_i)$ and $(T_{i,j} \leq \tau_{e,j})$ then
10: $U_{i,j} \leftarrow (1 - \lambda) p_j \mu_{i,j} - E_{i,j}^{\text{comp}}$
11: else
12: $U_{i,j} \leftarrow \lambda p_j \mu_{i,j} - E_{i,j}^{\text{comm}}$
13: end if
14: end if
15: if $s_j \notin T_{i,j}$ then
16: $T_{i,j} = (T_{i,j}^{\text{comp}} + T_{i,j}^{\text{comm}}) \mathbbm{1}_{i,j}$
17: if $(\mu_{i,j} \leq \mu_i)$ and $(T_{i,j} \leq \tau_{e,j})$ then
18: $U_{i,j} \leftarrow (1 - \lambda) p_j \mu_{i,j} - E_{i,j}^{\text{comp}}$
19: else
20: $U_{i,j} \leftarrow 0$
21: end if
22: end if
23: end for
24: end for
25: return $U$

B. Learning Process

1) Available Actions: To avoid selecting unmatched servers that cannot process such tasks as actions, and to improve the learning efficiency by reducing the time complexity, we need to clarify which set of actions are available in each state before selecting one as the action. Thus, we define the concept of available actions as below.

Definition IV.1. (Available Actions) Assume the agent is at state $\Omega_j$, $A^{\text{ava}}_{\Omega_j} = \{s_1, s_2, \ldots, s_n\}$ is the set of available actions, i.e., servers, that can process $t_j$ under the constraints of $C1-C4$, with $n \leq n$ being the number of available actions.

Since the transfer of states is a dynamic process and the states will affect each other, one of the most direct effects is that once an action is selected, its computing power will be reduced and therefore will constrain the action selection of the subsequent states. Therefore, we evaluate the available actions of each state one by one. Recall the discussions in Section IV-A3, we get an $n \times m$ rewards table $U$, and we can use $U_j = \{U_{1,j}, U_{2,j}, \ldots, U_{n,j}\}$ to denote all the possible rewards of $\Omega_j$. Generally speaking, $U_j$ has three types of value: positive, zero, and negative. We can select those actions with non-zero values as the available actions. However, this naive method can lead to some severe consequences. For example, one server is selected multiple times due to its non-zero value (this could happen in Q-learning, especially when the number of servers is small), but its capacity of available computing resources is not enough to handle all of these tasks even if the reward for each task is positive or optimal.

To address this challenge, we design a dynamic table to record the accumulative $\mu_{i,j}$, which can be denoted by $\mu_{i,j}^{acc}$. This table shares the same dimensions with the rewards table $U$ generated in Algorithm 1. Hence, we can design the following mechanism to get the available actions for each state.

First, naively select the actions with non-zero values from $U$ and initialize the values as zero to reduce the computational cost by avoiding going through the whole action space again. Let $A^{\text{ ava}}_{\Omega_j} = \{A^{\text{ ava}}_{\Omega_j,1}, A^{\text{ ava}}_{\Omega_j,2}, \ldots, A^{\text{ ava}}_{\Omega_j,n}\}$ be the set to contain the actions with non-zero values at state $\Omega_j$. In other words, the agent only needs to check the actions in $A^{\text{ ava}}_{\Omega_j}$ at state $\Omega_j$. Then, based on the selected actions, we can get a value of $\mu_{i,j}$. In this way, after multiple rounds, we can get the accumulative value $\mu_{i,j}^{acc}$. This is a dynamic process, which means that only those selected actions can contribute to their corresponding accumulative values. Besides, we need to consider the time constraint, ensuring that one server can compute the allocated tasks in time. We use $\tau_{e,j}$ to demonstrate the available computing time for the actions in $A^{\text{ ava}}_{\Omega_j}$, which is calculated via $\tau_{e,j} = (\mu_i - \mu_{i,j}^{acc})/f_i$. Next, we can decide whether an action $A_{\Omega_j}^{\text{ ava}}$ is available by the following rules: if $\mu_{i,j} < \mu_i$ and $\tau_{e,j} > \tau_{e,j}$, then $A_{\Omega_j}^{\text{ ava}}$ is considered as available; otherwise, $A_{\Omega_j}^{\text{ ava}}$ will be discarded from $A_{\Omega_j}^{\text{ ava}}$. In this way, we can get a set of available actions $A_{\Omega_j}^{\text{ ava}}$ at state $\Omega_j$, and the final action $A_j$ should be chosen from it according to the selection policy which will be discussed in Section IV-B2.

Algorithm 2 StateAvailableActions

Require: $A$, $\Omega_j$, $U$

Ensure: $A_j$

1: Initialize $A$, $A^{\text{ ava}}_{\Omega_j}$, $\mu_{i,j}^{acc}$
2: for $i \in \{1, \ldots, n\}$ do
3: if $U_{i,j} \neq 0$ then
4: Append $A_i$ into $A_{\Omega_j}^{\text{ ava}}$
5: end if
6: end for
7: Choose $A_{\Omega_j}^{\text{ ava}}$ from $A_{\Omega_j}^{\text{ ava}}$
8: $\tau_{e,j}^{\text{ ava}} \leftarrow (\mu_i - \mu_{i,j}^{acc})/f_i$
9: if $A_{\Omega_j}^{\text{ ava}} \in A_{\Omega_j}^{\text{ ava}}$ then
10: if $\mu_{i,j}^{acc} \geq \mu_i$ or ($\tau_{e,j}^{\text{ ava}} < \tau_{e,j}$) then
11: Remove $A_{\Omega_j}^{\text{ ava}}$ from $A_{\Omega_j}^{\text{ ava}}$
12: $A_j \leftarrow$ choose another action from $A_{\Omega_j}^{\text{ ava}}$
13: end if
14: else
15: Append $A_{\Omega_j}^{\text{ ava}}$ into $A_{\Omega_j}^{\text{ ava}}$
16: $A_j \leftarrow A_{\Omega_j}^{\text{ ava}}$
17: end if
18: $\mu_{i,j}^{acc} \leftarrow \mu_{i,j}^{acc} + \mu_{i,j}$
19: return $A_j$

The whole process of choosing available actions for a specific state is summarized in Algorithm 2. At the beginning, we initialize $A_{\Omega_j}^{\text{ ava}}$, $A_{\Omega_j}^{\text{ ava}}$, and $\mu_{i,j}^{acc}$ (Line 1), and we get the
non-zero values from \( \mathcal{U} \) (Lines 2-6). Next, we choose an action from \( A_{j}^{nz} \) and calculate its available computing time (Lines 7-8). Then, we can determine whether the selected action is available or not (Line 9-17), and we update \( \mu_{i,j}^{nza} \) (Line 18). Finally, we get the available action \( A_{j} \) at state \( \Omega_{j} \). Please note that the selection method in Lines 7 and 12 will be further discussed in Section IV-B2. In general, Algorithm 2 is the transformation of C1 and C2 in the MTA problem.

2) The Update of Q-table: The Q-table is applied during the learning process, which is used to facilitate the action selection. First, we generate an \( m \times n \) matrix as the Q-table \( Q(\Omega, A) \), and initialize its value with 0. Then, we need to update the Q-value at each state. Here we use the following equation to update Q-value:

\[
Q(\Omega_{j}, A_{i}) = Q(\Omega_{j}, A_{i}) + \alpha [U_{i,j} + \gamma \max Q(\Omega'_{j}, A_{j}^{nza}) - Q(\Omega_{j}, A_{i})],
\]

where \( Q(\Omega'_{j}, A_{j}^{nza}) \) means all the possible Q-value at next stage: \( 0 \leq \alpha \leq 1 \) is the learning rate and \( 0 \leq \gamma \leq 1 \) is the discount factor.

**Algorithm 3 Learning-based Algorithm for the MTA Problem**

**Require:** \( \Omega, A, K \)

**Ensure:** \( TS^* \)

1: Initialize \( Q(\Omega, A), k \)
2: \( \mathcal{U} \leftarrow \text{StateActionReward}(\Omega, A) \)
3: \( A_{j}^{nz} \leftarrow \text{actions with non-zero values in } \mathcal{U} \)
4: Generate a random value \( x \)
5: if \( x \leq \epsilon \) then
6: \( A_{j} \leftarrow \text{randomly selected from } A_{j}^{nz} \)
7: else
8: \( A_{j} \leftarrow \text{max } Q(\Omega_{j}, .) \)
9: end if
10: for \( k \in \{1, \cdots, K\} \) do
11: for \( j \in \{1, \cdots, m\} \) do
12: \( A_{j} \leftarrow \text{StateAvailableActions}(A, \Omega_{j}, \mathcal{U}) \)
13: \( Q(\Omega_{j}, A_{i}) \leftarrow Q(\Omega_{j}, A_{i}) + \alpha [U_{i,j} + \gamma \max Q(\Omega'_{j}, A_{j}^{nza}) - Q(\Omega_{j}, A_{i})] \)
14: end for
15: \( TS \leftarrow \text{the sum of all rewards in this episode} \)
16: end for
17: \( TS^* \leftarrow \text{max } TS \)
18: return \( TS^* \)

The agent selects the action based on the Q-value, and we adopt the \( \epsilon \)-greedy algorithm as the policy of action selection. The \( \epsilon \)-greedy strategy selects the action with the largest expected rewards most of the time, and the parameter \( \epsilon \) balances exploration and exploitation. We can use a large value of \( \epsilon \) to allow the agent to exploit what have been learned, and the agent will explore more actions that have not been learned with a smaller \( \epsilon \). The learning process requires multiple rounds so that the agent can learn more about the rewards and find the best solution. For each episode \( k \in \{1, 2, \cdots, K\} \) where \( K \) is the total episodes of learning, we let the agent go through all the states and find a solution, and then we compare all the solutions \( TS \) and choose the one \( TS^* \) with the highest sum of rewards as the optimal solution.

| TABLE I: Basic Parameter Settings |
|-----------------------------------|
| \( n = 20 \) | \( m = 50 \) | \( D_{k} = [200, 400] \) | \( \theta = 0.01 \) |
| \( p_{j} = [1, 10] \) | \( D_{j} = [10, 20] \) | \( \tau_{e,j} = [1, 100] \) | \( \gamma = 0.9 \) |
| \( f_{j} = [1, 10] \) | \( D_{t} = [5, 10] \) | \( G_{i} = [5, 10] \) | \( \alpha = 0.01 \) |
| \( \delta = 0.01 \) | \( \delta_{i} = [5, 10] \) | \( \epsilon = 0.3 \) | \( n_{k} = 300 \) |

Algorithm 3 is the detailed process of the learning-based solution for the MTA problem. We first initialize \( Q(\Omega, A), k \), and get \( U_{i,j} \) via Algorithm 1 (Lines 1-2). The \( \epsilon \)-greedy strategy is implemented to select the action (Lines 3-9). Then, the learning process will last until episode \( k \) reaches the predefined total number of episodes \( K \) (Lines 10-16), and the optimal solution will be obtained and returned (Line 17-18).

**C. Complexity Analysis**

The learning-based algorithm comprises three sub-algorithms, so the complexity analysis needs to take all of them into account. First, the computational complexity of Algorithm 1 is \( O(m \times n) \), and the complexity of Algorithm 2 is \( O(n) \). As for the time complexity of Algorithm 3, it is \( O(k \times n) \). Thus, in general, the time complexity of the learning-based solution should be \( O(m \times n) = O(k \times n) + O(n) + O(m \times n) \), which means that we can solve the MTA problem in polynomial time.

**V. EXPERIMENTAL EVALUATION**

**A. Experimental Setting**

In our experiments\(^1\), we consider a blockchain-based MEC system in Metaverse with 20 servers and 50 offloading tasks as the primary setting. We change relevant parameters to analyze the impacts of the numbers of servers and tasks on the total rewards and the time complexity, as well as the influence of Q-learning parameters on the convergence. For clarity, we summarize the basic parameter settings in Table I.

\(^1\)The code is available at: https://github.com/wzljerry/Blockchain-based-Edge-Resource-Sharing-for-Metaverse

**B. Experimental Results**

First, we design experiments to prove the effectiveness of our proposed learning-based algorithm for the MTA problem. To that aim, we compare the learning-based solution with two benchmark algorithms, i.e., the random allocation algorithm and the greedy-based algorithm. Specifically, The random allocation algorithm allocates each task to one server randomly; and the greedy-based algorithm selects the server with the maximum reward for each task. We run these three algorithms with different numbers of servers and tasks to obtain the comparison results as shown in Fig. 2. From Figs. 2(a) and 2(b), we can see that the learning-based algorithm can always outperform the other two algorithms with higher rewards. The random strategy fluctuates a lot due to the randomness of each selection. The greedy and learning algorithms, in contrast, perform more smoothly. Intuitively, more servers will not result in higher total rewards; however, more tasks would increase the overall revenue. But from our experimental
results, this trend does not hold. There are at least three reasons for it. First, C1 – C4 constrain the allocation decisions, resulting in dynamic decisions as the number of servers or tasks changes, which does not have a cumulative effect on rewards; second, although more tasks result in more rewards, they also increase the cost of computation and communication; and third, servers and tasks are heterogeneous with different parameters, exacerbating the results from the first two reasons.

![Image](a) The Number of the MEC Servers (b) The Number of Offloading Tasks

Fig. 2: The influences of the numbers of MEC servers and offloading tasks on the total rewards.

![Image](a) The Increase Percentage of Unit Price (b) The Increase Percentage of Data Size

Fig. 3: The influences of unit prices and data sizes of offloading tasks on the total rewards.

Then, we explore the influences of unit prices and data sizes of offloading tasks on the total rewards. We increase \( p_j \) and \( D_j \) with the percentage from 10% to 100%, and the results are shown in Fig. 3. It is clear that both the increase of \( p_j \) and \( D_j \) will affect the total rewards. When the unit prices are larger, the users will pay more to the system, so the total rewards will increase. And, if the data size of \( D_j \) is larger, then the CPU cycles required to process \( D_j \) will also be increased, and thus the total rewards will be higher.

![Image](a) The Learning Rate \( \alpha \) (b) The Discount Factor \( \gamma \)

Fig. 4: The convergence of the learning-based algorithm.

Last but not least, we mainly examine the convergence performance of the learning algorithm. Q-learning is determined by two main variables, i.e., \( \alpha \) and \( \gamma \). From Fig. 4(a), we can see that when \( \alpha \) is smaller, the current choice is more influenced by experience and lacks further exploration, so although it achieves a high payoff at the beginning, the convergence rate does not improve effectively. From Fig. 4(b), we can see that the larger the value of \( \gamma \), the better the convergence. This is because \( \alpha \) takes into account the effect of future rewards on current choices, so as \( \gamma \) becomes larger, it emphasizes rewards more and thus learns the optimal combinations more efficiently to speed up the convergence.

VI. CONCLUSION

In this paper, we establish a blockchain-based MEC platform for resource sharing and optimization to facilitate Metaverse applications. In particular, we design a task allocation scheme to assist our proposed system. To that aim, a learning-based algorithm is proposed to help the system make task allocation decisions in polynomial time. Numerous experiments prove that our proposed system and algorithms are efficient.

REFERENCES

[1] M. Xu, W. C. Ng, W. Y. B. Lim, J. Kang, Z. Xiong, D. Niyato, Q. Yang, X. S. Shen, and C. Miao, “A full dive into realizing the edge-enabled metaverse: Vision, enabling technologies, and challenges,” arXiv preprint arXiv:2203.05471, 2022.

[2] S. Mystakidis, “Metaverse,” Encyclopedia, vol. 2, no. 1, pp. 486–497, 2022.

[3] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile edge computing: A survey,” IEEE Internet of Things Journal, vol. 5, no. 1, pp. 450–465, 2017.

[4] S. A. Huda and S. Moh, “Survey on computation offloading in unenabled mobile edge computing,” Journal of Network and Computer Applications, p. 103341, 2022.

[5] S. Dhelim, T. Kechadi, L. Chen, N. Aung, H. Ning, and L. Atzori, “Survey on computation offloading in UAV-enabled mobile edge computing,” Journal of Network and Computer Applications, p. 103341, 2022.

[6] M. Xu, D. Niyato, J. Kang, Z. Xiong, C. Miao, and D. I. Kim, “Wireless edge-empowered metaverse: A learning-based incentive mechanism for virtual reality,” arXiv preprint arXiv:2111.03776, 2021.

[7] H. Sheng, S. Wang, Y. Zhang, D. Yu, X. Cheng, W. Lyu, and Z. Xiong, “Near-online tracking with co-occurrence constraints in blockchain-based edge computing,” IEEE Internet of Things Journal, vol. 8, no. 4, pp. 2193–2207, 2020.

[8] M. Zhao Feng, W. Xiao Chang, D. K. Jain, H. Khan, G. Hong min, and W. Zhen, “A blockchain-based trusted data management scheme in edge computing,” IEEE Transactions on Industrial Informatics, vol. 16, no. 3, pp. 2013–2019, 2020.

[9] M. Castro, B. Liskov et al., “Practical byzantine fault tolerance,” in OSDH, vol. 99, no. 1999, 1999, pp. 173–186.

[10] X. Xu, D. Zhu, X. Yang, S. Wang, L. Qi, and W. Dou, “Concurrent practical byzantine fault tolerance for integration of blockchain and supply chain,” ACM Transactions on Internet Technology (TOIT), vol. 21, no. 1, pp. 1–17, 2021.

[11] F. Izdemir, Z. İdeniş İğer et al., “Rivest-shamir-adleman algorithm,” in Partially Homomorphic Encryption. Springer, 2021, pp. 37–41.

[12] Z. Wang, Q. Hu, R. Li, M. Xu, and Z. Xiong, “Incentive mechanism design for joint resource allocation in blockchain-based federated learning,” arXiv preprint arXiv:2202.10938, 2022.

[13] X. Li and X. Zhang, “Multi-task allocation under time constraints in mobile crowdsensing,” IEEE Transactions on Mobile Computing, vol. 20, no. 4, pp. 1494–1510, 2019.

[14] K. Jiang, H. Zhou, D. Li, X. Liu, and S. Xu, “A q-learning based method for energy-efficient computation offloading in mobile edge computing,” in 2020 29th International Conference on Computer Communications and Networks (ICCCN). IEEE, 2020, pp. 1–7.

[15] W.-C. Chien, H.-Y. Weng, and C.-F. Lai, “Q-learning based collaborative cache allocation in mobile edge computing,” Future generation computer systems, vol. 102, pp. 603–610, 2020.