Hindawi
Security and Communication Networks
Volume 2021, Article ID 8800234, 15 pages
https://doi.org/10.1155/2021/8800234

Research Article

Latency-Aware Computation Offloading for 5G Networks in Edge Computing

Xianwei Li and Baoliu Ye

Hohai University, Information Department, School of Computer and Information, Nanjing 21106, China

Correspondence should be addressed to Xianwei Li; lixianwei@njxzc.edu.cn

Received 30 July 2021; Revised 27 August 2021; Accepted 4 September 2021; Published 22 September 2021

Academic Editor: Xuyun Zhang

With the development of Internet of Things, massive computation-intensive tasks are generated by mobile devices whose limited computing and storage capacity lead to poor quality of services. Edge computing, as an effective computing paradigm, was proposed for efficient and real-time data processing by providing computing resources at the edge of the network. The deployment of 5G promises to speed up data transmission but also further increases the tasks to be offloaded. However, how to transfer the data or tasks to the edge servers in 5G for processing with high response efficiency remains a challenge. In this paper, a latency-aware computation offloading method in 5G networks is proposed. Firstly, the latency and energy consumption models of edge computation offloading in 5G are defined. Then the fine-grained computation offloading method is employed to reduce the overall completion time of the tasks. The approach is further extended to solve the multiuser computation offloading problem. To verify the effectiveness of the proposed method, extensive simulation experiments are conducted. The results show that the proposed offloading method can effectively reduce the execution latency of the tasks.

1. Introduction

With the development of wireless communication technology and the Internet of Things (IoT), a variety of emerging applications, such as intelligent access control based on facial recognition, path planning, and virtual reality, meet the needs of people and provide great convenience [1, 2]. However, these applications are usually resource-hungry and delay-sensitive while the physical limitations and the computing power of the mobile devices cannot undertake such applications [3, 4]. Mobile cloud computing is seen as an effective solution to provide computing resources for resource-constrained mobile devices. By offloading computing-intensive tasks to resource-rich cloud data centers for execution, the computing capabilities of mobile devices can be greatly expanded [5, 6].

5G is a major leap in the development of mobile communications [7]. 5G networks deploy ultra-dense distributed networks in small cell infrastructures to provide continuous connectivity [8, 9]. However, this does not mean that the requests of many users can be satisfied at the same time, because most of the computing tasks of applications are deployed in centralized data centers for execution. With the explosive growth of IoT devices in the 5G era, there will be massive computing tasks to be migrated to cloud data centers for executing, causing extreme pressure for the Internet and bringing high network latency. At the same time, the long distance between mobile devices and cloud data centers will also cause unpredictable delays [10–12].

In 5G mobile edge computing, how to offload application services reasonably and content to the edge network is a key issue. Compared with traditional cloud data centers, edge servers are constructed by machines with limited resources and use appropriate strategies to offload services to edge servers [13–15]. In 5G, multiple heterogeneous edge servers can be deployed in the edge network to provide computing services to different users [16, 17]. As a new network paradigm, Software Defined Network (SDN) can realize the logical centralized control of distributed user equipment [18]. Each user equipment transmits its

task-related information to the SDN controller at the beginning that can take appropriate methods from a global perspective to determine where and when to perform these tasks belonging to different users [19, 20].

Considering the increasing complexity of IoT applications in the 5G era, the tasks of a single application are often composed of a series of subtasks. Initially, the task is decomposed into multiple subtasks to support multithreaded processing and improve the efficiency of task execution [15]. Most of the existing research treats tasks as a whole and ignores the connection of internal subtasks. Using subtasks as the unit of computation offloading and performing fine-grained computation offloading, the parallel processing of certain subtasks can be realized, thereby further reducing the delay of tasks [16, 21].

Based on the above observation, a latency-aware computation offloading method is proposed. Through analyzing the dependencies between subtasks, the method rationally arranges scheduling between subtasks and executes fine-grained computation offloading to solve the delay problem caused by increasing computing demands. Specifically, the main contributions of our work are as follows:

(i) Propose a latency-aware computation offloading method in 5G networks which effectively reduces the overall completion time of the tasks.

(ii) Sovle the multiuser computation offloading problem by extending fine-grained computation offloading method with designed algorithm.

(iii) Through extensive simulation experiments, our proposed offloading method can effectively reduce the execution latency of the tasks.

The rest of the paper is organized as follows: Section 2 describes related work. In Sections 3 and 4, the goal of reducing the delay of tasks is raised and the algorithms for computing offloading decision-making are proposed. Experiments are conducted in Section 5. In Section 6, we conclude this paper.

2. Related Work

In recent years, many IoT applications need to be offloaded to the cloud data center for processing. The emergence of 5G has accelerated this process and further increased the demand for computation offloading [22]. To solve this problem, edge computing, as an effective computing paradigm, is widely used in 5G networks for computing offloading. By providing computing and storage resources at the edge of the network, task waiting time is reduced and user experience is better [23–25].

Most of the current research on edge computing offloading focuses on optimizing certain specific goals through reasonable computing offloading strategies. Jararweh [26] proposed a framework based on edge computing, using the expanded computing power of edge computing and 5G to effectively manage and optimize the energy cloud system, while improving its reliability and safety. Li et al. [27] studied Mobile Edge Computing (MEC) of Unmanned Aerial Vehicle (UAV) and maximized energy efficiency of unmanned aerial vehicles to achieve the smallest UAV energy consumption by optimizing UAV trajectory, user transmission power, and computing load distribution. Merluzzi et al. [28] proposed an energy-saving algorithm for dynamic computing offloading in multiaccess edge computing scenarios, using limited block length and reliability constraints to consider Ultra Reliable Low-Latency Communication (URLLC). The proposed algorithm is based on stochastic optimization, which achieves the best balance between service delay and energy spent on mobile devices while ensuring the target probability of service interruption. Yang et al. [29] built a multi-UAV-assisted mobile edge computing system to provide computing offloading services for terrestrial IoT nodes with limited local computing capabilities. To balance the load of UAVs, a multi-UAV deployment mechanism based on Differential Evolution (DE) is proposed. It uses a near-optimal algorithm to solve the decisions of computation offloading. It guarantees coverage constraints and satisfies the IoT node Quality of Service (QoS) while achieving load balancing of these drones.

The 5G network based on edge computing has advantages in effectively offloading large-scale traffic, which is a promising architecture to alleviate the conflict between transmission performance and Quality of Experience (QoE). However, due to the mutual interference between wireless channels in the 5G network, it is difficult to provide satisfactory services to mobile users with existing solutions. Therefore, the optimal method of edge offloading in the 5G network has caused more and more research. Cao et al. [30] proposed a reliable and efficient multimedia service optimization framework. First, a reliable video service mechanism was constructed to help mobile users to distinguish between credible and economical services. Second, an effective wireless resource allocation strategy was established, using the Stackelberg model and other potential game models to achieve low-latency and energy-efficient video service optimization. In addition, Yang proposed a joint optimization scheme for task sharing and resource allocation in a 5G communication network based on edge computing. First, three modes for processing computationally intensive tasks are proposed, including local computing, fuzzy node computing, and edge node computing. For these three computing modes, the problem of computing task offloading is transformed into a joint optimization problem of time and energy consumption, and the authors used the interior point method to solve this problem. Yang [31] studied the computing offloading and subcarrier allocation problems in the MEC system based on multicarrier NOMA and used a deep reinforcement learning method for online computing offloading to solve this problem and greatly improve the computing speed of the MEC system.

For edge computing offloading, latency is a key indicator, and latency-aware edge computing offloading issues have gradually become a current research hotspot [32, 33]. To solve the problem how to generate the best mix of suitable microservices for applications in the mobile edge computing environment, Xia et al. [34] first attempted to study the Data, User, and Power Allocation problem in the edge environment and proposed a two-stage game theory decentralization algorithm to achieve the Nash equilibrium as the

solution, which maximizes the user's overall data rate. Harris et al. [35] defined the problems of virtual network function placement and distribution and provided algorithms with guaranteed performance to realize the placement of delay-sensitive services in appropriate network locations according to the specific needs and related requirements of each service. In response to the need for Mobile edge orchestrator (MEO) to expand capacity on many devices, Nguyen et al. [36] proposed a fuzzy-logic based MEO that separates tasks from mobile devices and maps them to the cloud servers and edge servers, reducing the delay of task processing. Specially, the fuzzy-based MEO was employed to make multi-criteria decision-making which selects the appropriate host to perform tasks by considering multiple parameters in the same framework and find the optimal task segmentation strategy.

Although there was some work dedicated to solving the optimization problem of edge computing offloading in 5G, there is still relatively little work on latency-aware edge computing offloading while network delay is a key requirement for some delay-sensitive programs. Therefore, the delay-aware edge computation offloading method takes delay as the main optimization goal and reduces the total delay of task execution as much as possible to meet the needs of delay-sensitive tasks in 5G.

3. Models and Problem Definition

The delay and energy consumption models of edge computation offloading in 5G network are analyzed in this part, followed by the problem definition. The main symbols used in this section with their descriptions are shown in Table 1.

3.1. The Delay Model. Due to different task migration strategies, the delay of completing the task is also different, so the time delay is computed separately according to different computation offloading methods. Figure 1 illustrates a system framework for edge computing. In this framework, we consider a scenario where "S edge servers as providers of computing resources cover N mobile devices" has been changed to "S edge servers, as providers of computing resources, cover N mobile devices." Each mobile device can execute the task locally according to the specific situation or upload the task to edge servers, but only one migration strategy can be selected for a task.

For tasks executed locally, since data transmission is not performed, the time delay includes only the local execution delay. For the subtask $t_{i,j}$ on the specific user equipment u_i, the calculation method of local execution delay is as follows:

$$T_{i,j}^{\text{local}} = \frac{c_{i,j}}{f_i}, \tag{1}$$

where f_i represents the computing capability of the user equipment u_i.

For tasks that need to be migrated to the edge server, the time delay is divided into three parts: transmission delay, execution delay, and queuing delay. The computation method for the transmission rate $r_{i,j}$ between a certain user equipment u_i and the edge server e_j is as follows:

$$r_{i,s} = B \log_2\left(1 + \frac{h_{i,s}p_i}{\sigma + \sum_{i'=1,a_i=a_i}^{N} h_{i',s}p_{i'}}\right), \tag{2}$$

where B is the channel bandwidth, $h_{i,s}$ represents the channel gain between user equipment i and edge server s, p_i represents the transmission power of user equipment u_i, σ represents the basic noise power of the transmission channel, and $\sum_{I'=1,a_i=a_{I'}}^{N} h_{I',s}p_{I'}$ represents the wireless interference caused by other user equipment that transmits tasks to e_j.

The transmission delay of sending the subtask $t_{i,j}$ from the user equipment u_i to the edge server e_s is as follows:

$$T_{i,j}^{\text{trans}} = d_{i,j} \cdot \frac{1}{r_{i,s}}. \tag{3}$$

Since the computing power of different edge servers is different, the execution time of tasks on different edge servers is also different. The execution delay of tasks $t_{i,j}$ is as follows:

$$T_{i,j}^{\text{exec}} = \frac{c_{i,j}}{f_i^s}, \tag{4}$$

where f_i^s represents the computing power of the edge server e_s.

The queuing delay of task $t_{i,j}$ depends not only on the execution completion time of the predecessor task, but also on the migration strategy of tasks on other devices. Therefore, the queuing delays obtained by various algorithms are different and since the start time and end time of each subtask are not fixed, they change according to the execution of the specific task. Two variables EST and EFT are defined for each subtask to represent the objective function. EST(j, s) represents the earliest execution time when the *j-th* subtask is offloaded to the edge server e_s while EFT(j, s) represents the earliest completion time of the *j-th* subtask in the edge server. The value 0 of EST is set for the first subtask which means that the subtask should be executed on the user's device.

The EST and EFT computations of other subtasks are computed recursively since the first subtask. To compute the EST of a subtask, the offloading strategy of all predecessor tasks of the subtask must have been determined and the computation must base on the completion time of the precursor task. Specifically, the EST of a subtask is as follows:

$$\text{EST}(j,s) = \max\left\{T_s^{\text{avail}}, \max\left(\text{EFT}(j') + C_{j,j'}\right)\right\} (j' \in \text{pred}(j)), \tag{5}$$

where T_s^{avail} represents the time that the edge server e_s is idle, and $C_{j,j'}$ represents the data transmission delay from the subtask $t_{i,j'}$ to $t_{i,j'}$, expressed as

$$C_{j,j_I} = \begin{cases} 0, & a_{i,j} = a_{i,j_I}, \\ T_{i,j}^{\text{trans}}, & \text{otherwise.} \end{cases} \tag{6}$$

After the scheduling strategy of the subtask is determined, the EFT of the subtask is computed according to the execution time of the task:

Table 1: Symbols and corresponding descriptions.

Symbol	Description
$c_{i,j}$	The number of CPU cycles required to execute $t_{i,j}$
$d_{i,j}$	Input data volume of subtask $t_{i,j}$
T_i^{local}	The execution delay of the subtask $t_{i,j}$ in the local execution
$T_{i,j}^{\text{trans}}$	Transmission delay of subtask $t_{i,j}$
$T_{i,j}^{\text{exec}}$	Execution delay of subtask $t_{i,j}$ after migration
f_i	The computing power of the user's device ui
$r_{i,j}$	Transmission rate between u_i and e_j
B	The bandwidth of the transmission channel
$h_{i,j}$	Channel gain between user equipment i and edge server j
$E_{i,j}^{\text{local}}$	Execution energy consumption of subtask $t_{i,j}$ executed locally
$E_{i,j}^{\text{trans}}$	Transmission energy consumption of subtasks
EST	The earliest start time when the subtask gets the execution
EFT	Earliest completion time of subtask execution

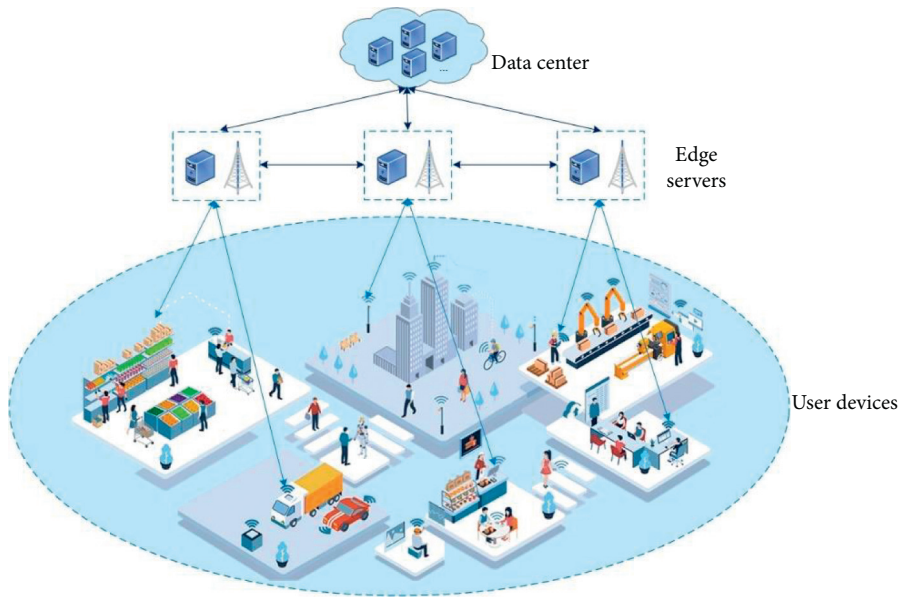

Figure 1: The architecture of computation offloading for 5G networks in edge computing.

$$\text{EFT}(j) = \begin{cases} \text{EST}(j,0) + T_{i,j}^{\text{local}}, & t_{i,j} \text{are executed locally,} \\ \text{EST}(j,s) + T_{i,j}^{\text{exec}}, & \text{otherwise.} \end{cases} \tag{7}$$

For a task t_i, its completion time is determined by the EFT of its last completed task. The computation method is as follows:

$$T_i^{\text{finish}} = \text{EFT}(\text{last}) + T_{\text{last}}^{\text{local}}. \tag{8}$$

Since the last task is usually to collect and process computation results, it is generally executed locally on the user device, and $T_{\text{last}}^{\text{local}}$ is used to represent the execution time of the last task, so as to obtain the end time of the entire task.

3.2. The Energy Consumption Model. The energy consumption of the user equipment u_i mainly includes two parts, which are the execution energy consumption caused by the execution of tasks on the user equipment and the transmission energy consumption of offloading the tasks to the edge server.

If the task $t_{i,j}$ decides to be executed locally, the execution energy consumption is as follows:

$$E_{i,j}^{\text{local}} = \delta_i \cdot c_{i,j}, \tag{9}$$

where δ_i is the energy consumption of per unit CPU cycle of the user equipment u_i.

If task $t_{i,j}$ decides to migrate to the edge server e_s for execution, the corresponding transmission energy consumption is as follows:

$$E_{i,j}^{\text{trans}} = p_i \cdot d_{i,j} \cdot \frac{1}{r_{i,j}}. \tag{10}$$

The total energy consumption of task t_i is determined according to the different migration strategies of each subtask. Set a binary variable flag_j to indicate whether the j-th subtask is to be migrated. The computation method is as follows:

$$\text{flag}_j = \begin{cases} 0, & t_{i,j} \text{ are executed locally,} \\ 1, & t_{i,j} \text{ are executed in edge servers.} \end{cases} \tag{11}$$

The total energy consumption of task t_i is as follows:

$$E_i^{\text{total}} = \sum_{j=1}^{M} \text{flag}_j \cdot E_{i,j}^{\text{trans}} + \left(1 - \text{flag}_j\right) \cdot E_{i,j}^{\text{local}}. \tag{12}$$

3.3. Problem Definition. This research aims to find a set of edge computing offloading strategies to minimize the time delay while meeting the constraints of user equipment on energy consumption. The problem is formalized as

$$\min \sum_{i=1}^{N} T_i^{\text{finish}}, \tag{13}$$

$$\text{s.t. } E_i^{\text{total}} \le L_i, \tag{14}$$

where L_i represents the battery power of the user equipment u_i. Equation (14) indicates that the energy consumption of the user equipment to perform tasks cannot exceed the power of the user equipment.

4. Algorithm Design

In this section, based on the problem of computing offloading in the edge network proposed in the previous chapter, a delay-aware optimization algorithm is proposed. Because the execution effect of the entire task depends on the scheduling strategy of each subtask and the dependencies between the subtasks, a method for selecting the optimal offloading strategy for each subtask is proposed, and then the optimal scheduling strategy for the entire task is obtained. To better cope with the computing offloading needs of multiple users, the proposed computing offloading strategy selection algorithm is further expanded, so that it can solve the problem of computing offloading decision-making in a multitasking environment. At the same time, to cope with the possible delay in the unknown network environment, the proposed method is further improved.

4.1. The Subtask Selection Algorithm. How to choose the most suitable subtasks for computational offloading requires solving the following two problems. First, because different subtasks have different benefits for computing offloading, how to determine which subtasks can be offloaded from the topological graph of the computing task. Second, what kind of subtask offloading combination can provide greater potential performance. Therefore, it is necessary to analyze the topological structure diagram of the subtasks to obtain the opportunity for computing offloading. Due to the interdependence between different subtasks, the first thing to be solved is to sort each subtask to determine the order of scheduling. At the same time, there are multiple parallel subtasks in a task. How to sort these parallel subtasks and determine the scheduling priority also has a certain impact on reducing the delay of the entire task.

Using $d_{a,b}$ represents the amount of data transmitted from subtask $t_{i,a}$ to subtask $t_{i,b}$, and the communication delay $\omega_{a,b}$ between edges (a, b) can be expressed as

$$\omega_{a,b} = \begin{cases} d_{a,b} \cdot \dfrac{1}{r_{a,b}}, & t_a \text{ and } t_b \text{ scheduling strategies are different,} \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

Obviously, when two consecutive subtasks are executed at the same place, there is no need for data transmission between two subtasks, so the communication delay is zero. Therefore, the mean value of the communication delay between the two subtasks can be expressed as

$$\overline{\omega}_{a,b} = d_{a,b} \cdot \frac{1}{2r_{i,a}}. \tag{16}$$

Similarly, the average execution delay of a subtask $t_{i,j}$ can be expressed as

$$\overline{T}_{i,j} = \frac{T_{i,j}^{\text{exec}} + T_{i,j}^{\text{local}}}{2}. \tag{17}$$

It is the average of the delays of local execution and migration to edge server execution.

According to these two mean values, the computing method of scheduling priority for a certain subtask $t_{i,j}$ is defined, which is specifically expressed as

$$\text{prior}(j) = \overline{T}_{i,j} + \max\left(\overline{\omega}_{j,j'} + \text{prior}\left(j'\right)\right)\left(j' \in \text{pred}(j)\right), \tag{18}$$

where pred(j) is the predecessor subtask set of $t_{i,j}$. prior(j) represents the scheduling priority of subtask $t_{i,j}$. Obviously, the value of priority(0) for the first subtask of the task is 0. The lower the priority value, the higher the scheduling priority of the subtask.

After determining the scheduling sequence, to select the optimal computation offloading method for each subtask, an algorithm for shortest time-to-completion first of the subtask is proposed which chooses a suitable offloading strategy based on computing volume and its predecessor subtask. Sort from high to low according to the previously computed scheduling priority and select the subtask with the highest scheduling priority among the unscheduled subtasks to process. For each subtask, the EFT executed locally and the EFT offloaded to the edge server according to (7) are computed firstly, and the delay of two schemes and the energy consumption constraint of the user equipment at this time are compared to select the optimal offloading strategy for the subtask.

The Algorithm 1 describes the specific process of selecting a subtask scheduling strategy. The EFP first inputs the directed acyclic graph representing the entire task and the relevant parameters needed to compute the EFT and then computes the values of scheduling priority of all the input subtasks. After all the prior values are obtained, the prior values are sorted in a nonincreasing manner. At this time,

the task with the lowest prior value will be scheduled first (line 2–5). The decision-making process of the specific offloading strategy for a single subtask is shown (line 6–14). By looping through each subtask, the earliest end time EFT of the local execution task and the offloading task to the edge server is computed for each task and according to the value of the two EFTs, decide whether to offload to the edge server for the subtask. At the same time, it is necessary to limit the number of subtasks executed locally to meet the energy consumption constraints of user equipment. The loop is ended when the migration strategy is determined for each subtask, and the output result at this time is the computation offloading strategy of task t_i.

4.2. The Offloading Strategy Selection Algorithm. In 5G network, multiple users often need to offload delay-sensitive tasks to the edge server at the same time to obtain computing resources of the edge server to improve task execution efficiency. The shortest completion time priority algorithm proposed in Section 4.1 is only designed to select the best offloading strategy for a single task. To better cope with the computing offloading needs of multiple users, the computing offloading strategy selection algorithm is further expanded, so that it can solve the problem of computing offloading decision-making in a multitasking environment.

First, because there are multiple edge servers which provide computing resources in a multitasking environment, it is necessary to make decisions on each edge server to select the migration strategy with the least delay. Secondly, in the migration process of multiple users, signal interference will also be brought to the data transmission channel, and a decision must be made in consideration of the interference between users. Considering the above problems, the EFP algorithm is further extended to make it possible to solve the problem of edge computing offloading in a multi-user environment.

For a specific scheduling strategy of a task t_i, specifically for a task with j subtasks, of_i = $\{a_1, a_2, \ldots, a_j\}$, where a_n represents the offloading strategy corresponding to the n-th subtask, if $a_n = 0$, the subtask will be executed locally; otherwise, it will be offloaded to the corresponding edge server. For users who need to compute offloading at the same time, list all possible migration strategies for all users, and then compute the delay currently for each possible migration strategy, and then compare to get the optimal offloading strategy. For a specific migration strategy, first, according to the determined strategy, record the number of tasks that will be offloaded to an edge service, and update some variables that change with the environment during the offloading process, such as the task's transmission rate $r_{i,s.}$. After that, the EFP algorithm proposed in Section 4.1 is called. For a user's task, first the scheduling priority of the subtasks within each task is computed, and then the offloading strategy of the subtasks is determined according to the priority to place the task in local or offloading tasks to edge servers to achieve lower latency. After computing the delay of each subtask, the total delay for a computing offloading strategy can be obtained. After comparing the experiments of all feasible offloading strategies, the computing offloading strategy with the least delay is selected as the output of the algorithm.

Algorithm 2, referred to as MEFP, describes the specific process of edge computing offloading decision-making in a multiuser environment. Firstly, enumerate all possible migration strategies according to the set U of all users to be offloaded at a certain time. Then by using a set to store tasks to be migrated to the same edge server in each migration strategy, all possible migration strategies that convenient to update the transmission interference between different tasks are looped through. Then for each user's specific tasks, the transmission rate is updated according to the network status, and the EFP algorithm is called to decide whether to migrate (line 3–10). After the decision is completed, the total task execution delay under the offloading strategy is obtained. After traversing all possible migration strategies, the migration strategy with the smallest total delay is selected and output as the result (line 11–13).

4.3. The Shortest Completion Time Priority Algorithm. The offloading strategy selection algorithm proposed in Section 4.2 provides the most optimized computing offloading strategy in a multiuser environment. However, when limited resources lead to high resource contention rate, some unreasonable decisions are made due to lack of overall network information. For example, there are many tasks waiting to be executed on the edge server at the same time while the client is not aware and still migrates tasks to the edge servers. Due to the long queuing delay, the execution delay of the entire task may be longer than executed locally, which occurs more frequently when the computing resources are insufficient.

To solve such problems, based on the Carrier Sense Multiple Access (CSMA) used in computer networks, a multiuser shortest completion time priority algorithm in a resource contention environment is proposed. In the CSMA algorithm, instead of directly sending data packets when detecting that the channel is idle, the sender refuses to send with a certain probability to avoid conflicts. Similar strategies are adopted to avoid conflicts in computing migration requirements when multiple tasks compete for edge server computing resources. After the comparison of execution time of processing task locally and migrating task to the edge server, the task migration is rejected with a certain probability. Because executing the task locally can prevent multiple tasks from waiting for the computing resources, reducing queuing delay and task execution delay.

The computation method of the transfer probability possibility(j) of subtask $t_{i,j}$ can be expressed as

$$\text{possibility}(j) = \frac{T_{i,j}^{\text{local}} - \left(T_{i,j}^{\text{exec}} + T_{i,j}^{\text{trans}}\right)}{T_{i,j}^{\text{queue}} + \tau}, \tag{19}$$

where $T_{i,j}^{\text{queue}}$ represents the waiting time at the edge server, τ represents a small time constant.

When the difference between the delay of local execution and the delay of task migration execution is larger than the

Inputs: $G = (T, DP)$, M, $r_{i,j}$, f_i, f_i^s, L_i
Output: Decision (i)
(1) $i = 0$
(2) **for** $i = 0$ **to** M **do**
(3) Compute the prior values of each subtask $t_{i,j}$ according to formula (17)
(4) **end for**
(5) non-increasing sorting of the prior values of all subtasks
(6) **while existing subtasks that have not confirmed scheduling strategies**
(7) Select the subtask $t_{i,j}$ with the highest priority
(8) Computing the EFT(j) of the task according to formula (7)
(9) **if** $\text{EFT}(j)_{\text{local}} < \text{EFT}(j)_{\text{edge}}$ **and** $E_i^{\text{totals}} < L_i$
(10) The subtask $t_{i,j}$ still executed locally
(11) **else**
(12) Offload $t_{i,j}$ to edge server
(13) **end if**
(14) **end while**

Algorithm 1: Shortest time-to-completion first EFP.

Inputs: E, U, $G_i = (T_i, DP_i)$, M, $r_{i,s}$, f_i, L_i
Output: Optimal offloading strategy $of_{\min}$ for multiusers
(1) List all optional offloading strategies $\text{OF} = \{of_1, of_2, \ldots, of_N\}$
(2) **for** of_i in OF **do**
(3) **for** a_j in of_i **do**
(4) **for** e_k in E **do**
(5) Create a set U_k for each users with $a_j = e_k$
(6) Update the transfer rate $r_{i,s}$ of each tasks
(7) **end for**
(8) **for** t_i in U_k **do**
(9) Call EFP algorithm
(10) **end for**
(11) **end for**
(12) Compute the total execution delay T_i^{finish} for offloading strategy of_i
(13) **end for**
(14) Select the offloading strategy $of_{\min}$ with minimal execution delay

Algorithm 2: Shortest time-to-completion first with multiusers MEFP.

waiting delay, the migration probability value at this time is at most 1 when the task must be offloaded to the edge server. On the contrary, when the local execution delay is less than the execution delay of the task migration, the migration probability value is 0 when the task is executed locally.

The migration probability of a task changes linearly with the waiting delay and the difference between the local execution delay and the migration execution delay. The larger the waiting delay of the last execution, the more congested the network conditions at this time, at which case the probability of task being executed locally is larger than being executed in edge servers. When the waiting delay is small, the probability of the task being computed and offloaded is greater. When the computing task is simple and the performance difference between the local and offloading to the edge server is not big, the computing task will have a greater probability to be executed locally to avoid the situation where multiple tasks are waiting to be executed at the edge server at the same time. When the computing task is more complex and the performance on the edge server is significantly better than the local execution, the computing task will be offloaded to the edge server. For the subtasks that use the EFP algorithm to determine the offloading strategy and need to perform computing offloading, it is necessary to further determine whether the task should be offloaded to the edge server according to the calculated probability.

Algorithm 3 describes the execution process of the multiuser shortest completion time priority algorithm in a resource contention environment. All possible migration strategies are enumerated and the EFP algorithm is employed to make decisions. Then the probability of selecting migration is computed at this time (line 1–8). If the random number generated is greater than the probability, the migration is rejected, and the task is executed locally. Otherwise, the task is still offloaded to the edge server (line 9–17). At the end of the algorithm, according to the delay of

Inputs: E, U, $G_i = (T_i, \mathrm{DP}_i)$, M, $r_{i,s}$, f_i, L_i,
Output: Optimal offloading strategy $\mathrm{of}_{\min}$ for multiusers
(1) List all optional offloading strategies $\mathrm{OF} = \{\mathrm{of}_1, \mathrm{of}_2, \ldots, \mathrm{of}_N\}$
(2) **for** of_i in OF **do**
(3) **for** a_j in of_i **do**
(4) **for** e_k in E **do**
(5) Create a set U_k for each user with $a_j = e_k$
(6) **for** t_i in U_k **do**
(7) Update the transfer rate $r_{i,s}$ of each tasks
(8) Call EFP algorithm
(9) **if** Decision(i, j) ! = 0
(10) Compute migration probability possibility(j)
(11) random = Random(0, 1)
(12) **if** random ≥ possibility(j) **and** < L_i
(13) Execute $t_{i,j}$ locally
(14) **else**
(15) Offload $t_{i,j}$ to edge server
(16) **end if**
(17) **end if**
(18) **end for**
(19) **end for**
(20) **end for**
(21) Compute the total execution delay T_i^{finish} for computing offloading strategy of_i
(22) **end for**
(23) Select the offloading strategy $of_{\min}$ with minimal execution delay

Algorithm 3: Shortest Time-to-Completion First with multiusers under low resource p-MEFP.

the corresponding computing offloading strategy, the offloading strategy with the smallest delay is obtained as the output result for all users who seek to computation offloading (line 20–22).

5. Experiment Evaluation

A series of experiments to simulate the process of multiuser edge computing offloading in 5G are carried out to verify the effectiveness of the proposed method. Firstly, experimental configuration like parameter settings is introduced, and then comparison schemes are selected to simulate the environment under different number of tasks and edge servers. Through comparison, the advantages of the proposed p-MEFP algorithm are shown obviously.

5.1. Experimental Configuration. The experiment simulates an environment with multiple edge servers that can provide computing resources at the same time, and there are multiple user devices randomly distributed around these edge servers in the network environment, and each user device has a set of tasks that consist of several subtasks; a single-edge server can perform multiple subtasks at the same time according to its own computing capabilities. To verify the feasibility of the proposed method for multiple tasks, the directed acyclic graph of the task is randomly generated within a certain range. The size of each subtask is randomly generated in [50 KB, 1000 KB], and the required CPU cycles vary randomly from 50 M cycles to 1000 M cycles. The specific parameters and corresponding values in the experiment are listed in Table 2.

To achieve comparison, another two-edge computing offloading algorithms implemented are introduced as follows:

(1) Benchmark: For each subtask in the task, according to the order of execution, all tasks are migrated to the edge server for execution, the task is not executed locally, and finally the execution structure of the task is transmitted back to the user device.
(2) CEFO1 [38] is an SDWN-based edge computing offloading method. Through the task data uploaded by each user device, the SDWN central controller determines the specific migration strategy for each task. First, enumerate all the optional offloading decisions. For each offloading scheme, the task graph of users which are offloaded to the same server is regarded as an integrated DAG graph through the combination of graphs, and the delay of each different offloading scheme is computed, and finally the offloading scheme with the minimum waiting time is selected as the offloading strategy.

5.2. Experiments Results. In this section, the performance of the proposed p-MEFP algorithm and the other two comparison algorithms Benchmark and CEFO are compared in detail from different user numbers and different edge servers, showing the effectiveness of the three methods in reducing execution delays. At the same time, compare the effectiveness of the p-MEFP algorithm for different task types. The experimental results are shown in Figures 1–9.

Table 2: Parameters and values.

Parameters	Values
Basic noise power of the transmission channel σ[40]	100 dBm
Task transmission power p[40]	150 mW
CPU frequency of edge server f_i^s[40]	20 GHZ
CPU frequency of user equipment f_i [40]	10 GHZ
Task execution power [37]	650 mW

5.2.1. Performance under Different Number of Users. This part compares the average task delay of each method after the simulation experiment of Benchmark, CEFO and p-MEFP under different resource contention environments. By changing the layout of the number of edge servers in the network environment and adjusting the capacity of each edge server, the network environment is set to a high contention environment (the number of edge servers is 2, and each edge server can perform at most 1 task), medium contention environment (the number of edge servers is 5, and each edge server can perform up to 2 tasks), and low contention environment (the number of edge servers is 8, and each edge server can perform up to 4 tasks).

Figure 2 shows the comparison of the average task queuing delay of different methods in a high resource contention environment. In an environment of high resource contention, as the number of tasks increases, the queuing delay is increasing rapidly, and the queuing delay gap of the method is obvious. When the number of tasks is 30, the maximum difference is 60 ms, and when the number of tasks is 50, the average queuing delay gap is up to 100 ms, indicating that the p-MEFP method can reduce the queuing delay of the task well.

Figure 3 shows the average task delay of the three algorithms under different number of tasks in a high contention environment. In a high contention environment, the number of edge servers that can provide computing resources is relatively small, and the use of edge servers for tasks is more obvious. It can be reflected from the figure that when the number of tasks is small, the execution effect of the three methods is similar, and the p-MEFP method is only slightly better. With the continuous increase in the number of tasks, the situation of users competing for edge servers becomes more and more serious, and the queuing delay accounts for an increasing proportion of the total delay. As the gap in queuing delay becomes larger, the average task delay difference of the three methods gradually becomes larger. When the number of tasks is 10, the difference between optimal and worst performance is only 10 ms. When the number of tasks is 30, the difference is 40 ms, but when the number of tasks is 50, the average delay of p-MEFP is reduced by nearly 70 ms compared to CEFO and is 80 ms less than Benchmark. It is extremely effective in reducing delay, and it significantly reduces execution delay. Through comparison, as the number of users continues to increase, the performance of p-MEFP gets better, which shows the effectiveness of p-MEFP in reducing task delay in a high resource contention environment. At the same time, with the increase in the number of tasks, the increase in delay is very fast. For every 10 additional tasks in p-MEFP, the increase in delay is within 100 ms, while for Benchmark, the increase in delay even reaches nearly 130 ms. Finally, when the number of tasks is 50, the delay of p-MEFP is 383 ms, while the delay of the other two methods is about 450 ms. In comparison, p-MEFP greatly improves the execution effect of the task and reduces the task delay. Comparing Figures 1 and 2, the largest proportion of the task delay at this time is the queuing delay, and p-MEFP can greatly reduce the queuing delay, and thus has a better delay performance.

Figure 4 shows the comparison of the average queuing delay of different methods in the medium resource contention environment. Initially, the queuing delay of the three methods is similar. When the number of tasks reaches 40 and even more, p-MEFP can reduce the queuing delay of nearly 20 ms and 30 ms compared with the other two respectively and has a great advantage in reducing the queuing delay. It proves that our method can achieve good results under the pressure of a large number of tasks.

Figure 5 shows the effectiveness of Benchmark, CEFO and p-MEFP in reducing the average task delay in a medium resource contention environment. In the case of medium resource contention, when the number of tasks is small, the difference between the three is only 6 ms, and as the number of tasks continues to increase, the difference gradually becomes larger, but the largest difference is only about 20 ms, but in all scale tasks, p-MEFP still has certain advantages. As the number of tasks continues to increase, the advantages of p-MEFP are becoming more and more obvious. In terms of the value of delay, the execution delay of tasks in a medium resource contention environment is significantly less than that in a high resource contention environment, and as the number of tasks increases, the rate of increase in delay is relatively stable, and the increase rate remains within 40 ms. And at this time, the queuing delay still accounts for a large proportion of the total execution delay of the task, so the reduction of the queuing delay can still greatly improve the execution effect of the task and reduce the execution delay of the task.

Figure 6 shows the comparison of the effectiveness of the three methods in reducing task queuing delay in a low resource contention environment. When the computing resources in the environment are abundant because there is more space in the edge server, it can accommodate more tasks to be executed at the same time. Currently, the queuing delay accounts for a relatively small proportion of the total delay and the impact of different migration strategies on the delay Smaller. For queuing delay, p-MEFP has no advantage in queuing delay due to the large number of idle edge services at the beginning, but the difference is also about 1 ms. As the number of tasks continues to increase, the

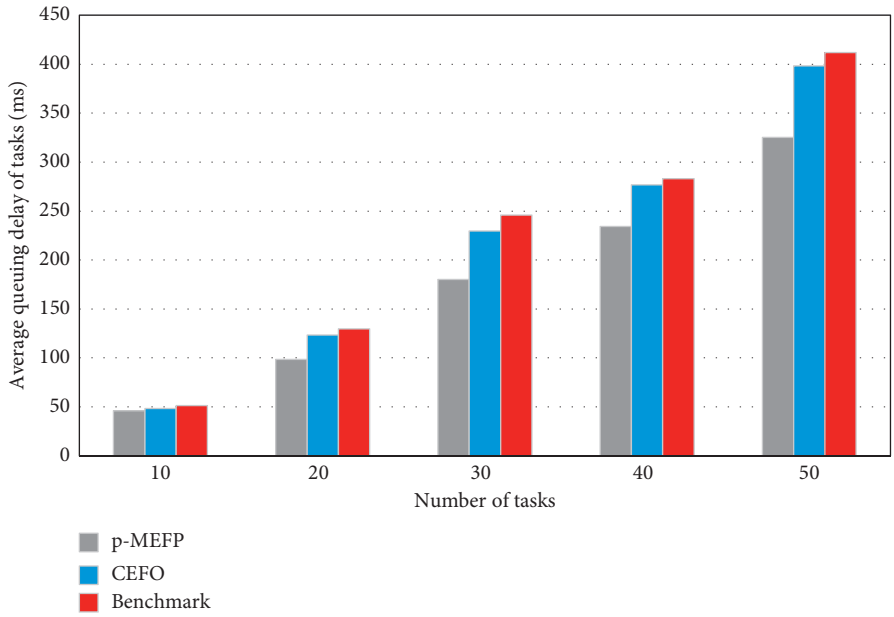

Figure 2: Comparison of the average queuing delay in a high resource contention environment.

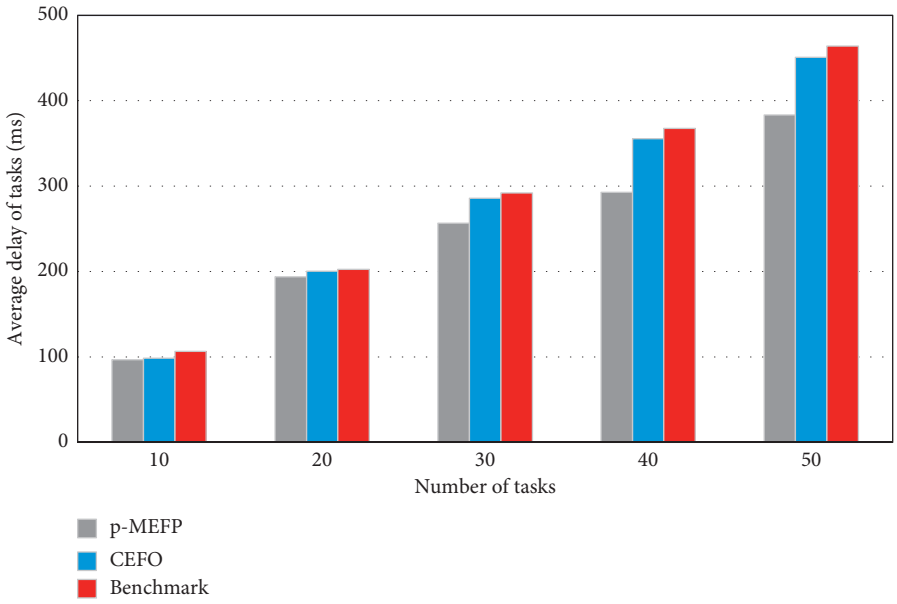

Figure 3: Comparison of the average delay in a high resource contention environment.

average queuing delay is significantly lower than the other two methods where tasks use the p-MEFP method.

Figure 7 is a comparison of the effectiveness of the three methods in reducing the average total delay in a low resource contention environment. Obviously, as the number of tasks increases, the average delay of tasks increases, which is caused by a large number of tasks queuing at the edge. When the number of tasks is low, p-MEFP can reduce task delay, but has no obvious advantage compared to other methods. Because in the case of sufficient resources and few tasks, the delay will be small. As the number of tasks increases, the effectiveness of p-MEFP gradually exceeds the other two methods.

5.2.2. Performance under Different Number of Edge Servers. This part compares the average task delay of the three algorithms of Benchmark, CEFO, and p-MEFP under different edge server numbers. The comparison of the average task delay of Benchmark, CEFP, and p-MEFP with different edge server numbers is shown in Figure 8.

When the number of edge servers is 2, the delay of p-MEFP is less than 400 ms, while the delays of the other two comparison methods are more than 450 ms. The effectiveness of p-MEFP is more obvious. It can reduce the delay of nearly 100 ms compared with Benchmark and nearly 50 ms compared with CEFO. When the number of edge servers is small, p-MEFP has a 30–50 ms advantage over the other two

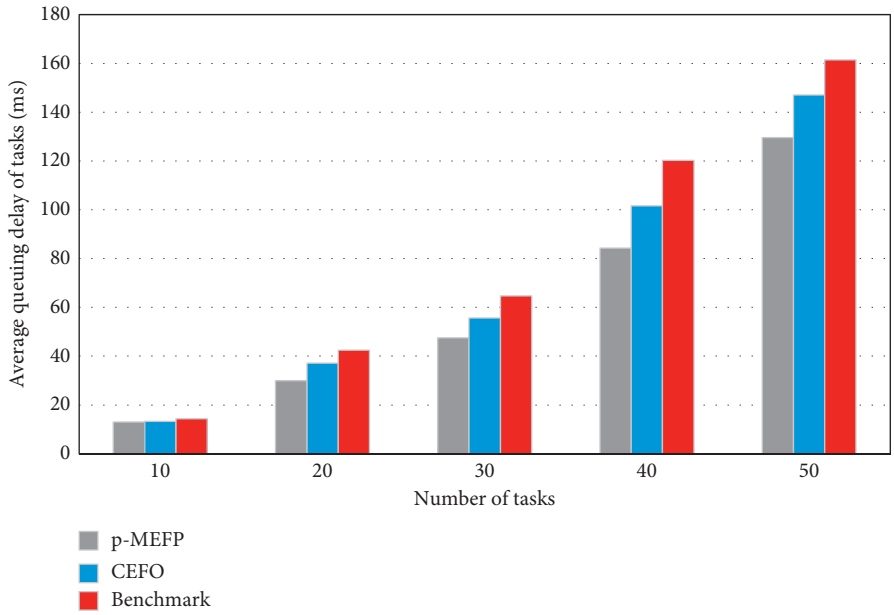

Figure 4: Comparison of the average queuing delay in a medium resource contention environment.

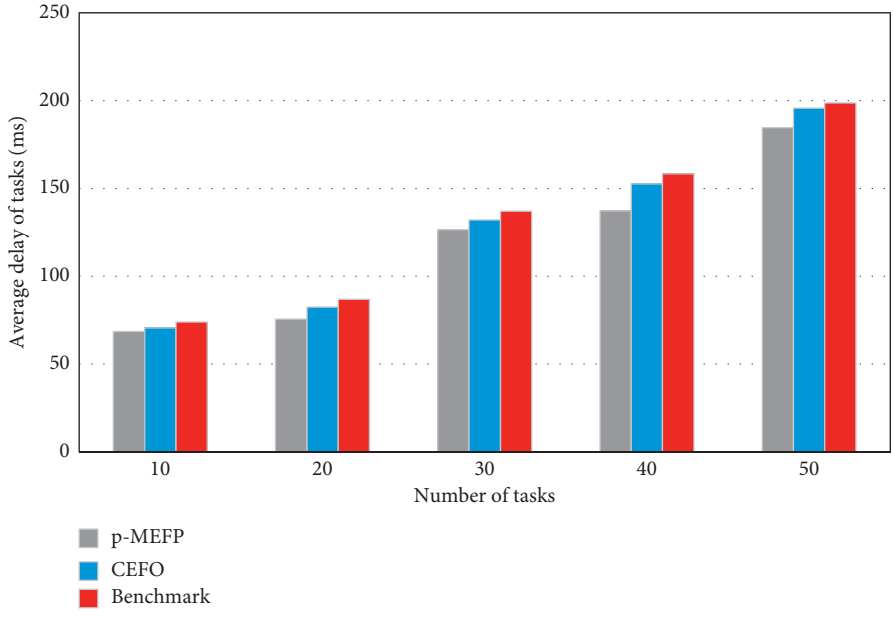

Figure 5: Comparison of the average delay in a medium resource contention environment.

methods. With the increase in the number of edge servers, the advantages of p-MEFP gradually decrease, but the overall p-MEFP has a certain optimization effect compared with the other two methods, which can reduce the average delay of task execution. When the number of edge servers is 5, the task execution delay of p-MEFP is 17 ms and 22 ms less than the other two methods, respectively. For the task execution delay of less than 200 ms at this time, the delay reduction effect is still obvious. After the number of edge servers is further increased to 8, the computing resources are sufficient at this time, and the requirements of computation offloading can be fully met, which is nearly 300 ms lower than when the number of edge servers is 2. The average task execution delay obtained by several comparison methods is not much different, and they are all reduced to about 120 ms. The p-MEFP method only reduces the execution delay by about 7 ms compared with other methods.

On the whole, p-MEFP still maintains its effectiveness in reducing latency, and it can be seen that the number of edge servers has a great impact on the latency of task execution.

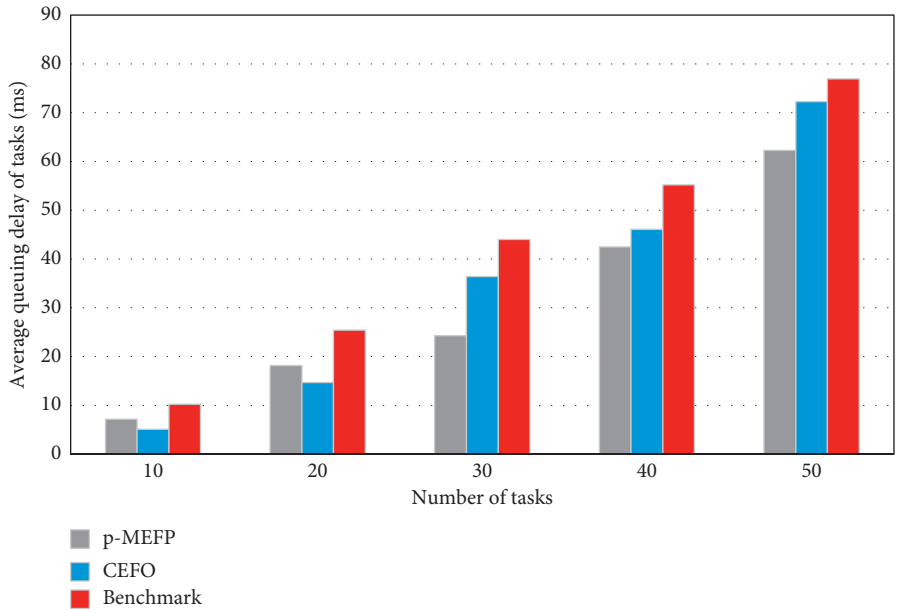

Figure 6: Comparison of the average queuing delay in a low resource contention environment.

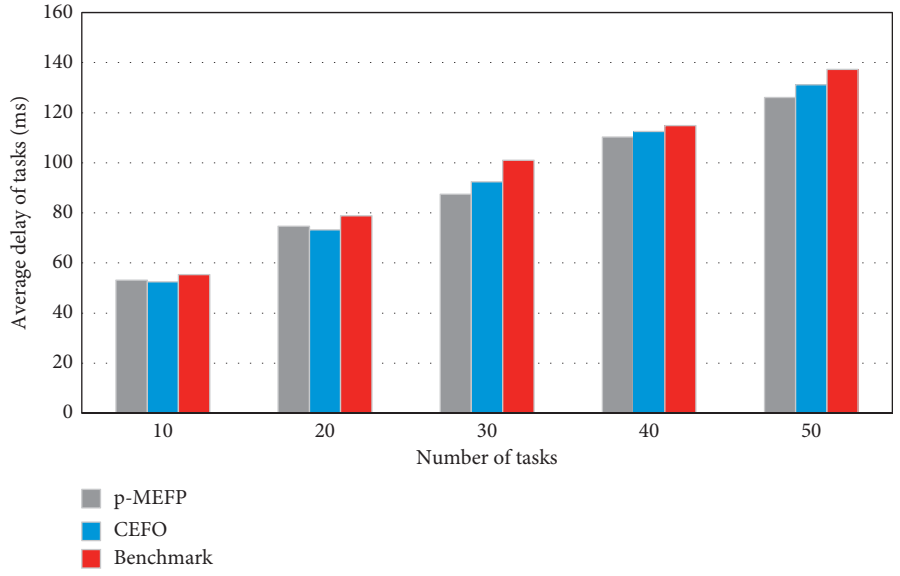

Figure 7: Comparison of the average delay in a low resource contention environment.

With the continuous increase of edge servers, the average execution latency of tasks will be reduced to one-third of the original.

5.2.3. Performance of Directed Acyclic Graphs for Different Tasks. Since the tasks discussed in this article may be decomposed into multiple subtasks, and the parallelism of the subtasks will have a certain impact on the execution effect of the task, in this section, experiments are carried out on different task directed acyclic graphs to compare the differences that the parallelism of the subtasks on the task execution effect. Due to the specific discussion of the directed acyclic graph of the task, five specific tasks with inconsistent parallelism were selected for experiments. The directed acyclic graph of the five tasks is shown in Figure 9. Task type 1 is the serial execution of five subtasks, and each subtask must wait for the completion of its predecessor task. Subtask 2 and subtask 3 of task type 2 can be executed in parallel, and subtask 4 can be executed only after they are all completed. Subtask 2, subtask 3, and subtask 4 of task type 3 can all be executed in parallel, and subtask 5 can only be executed after all three subtasks are completed. Task type 4 and task type 5 are similar; in that

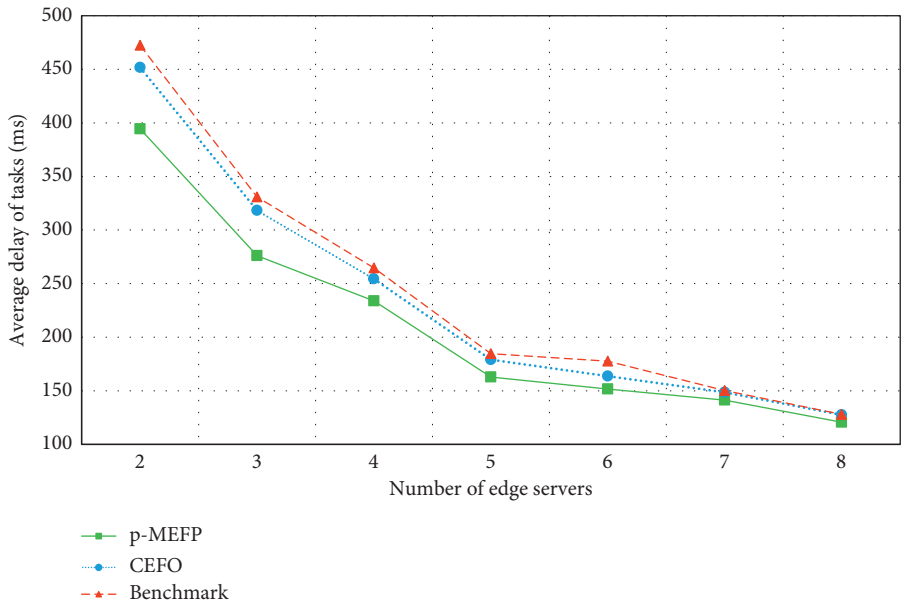

Figure 8: Comparison of the average delay of three algorithm with different number of edge servers.

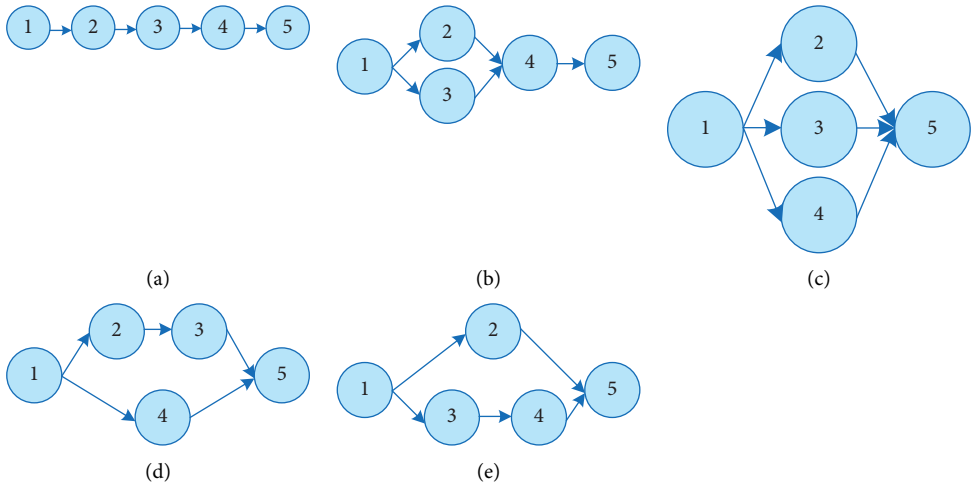

Figure 9: Directed acyclic graph participating in computing offloading task. (a) Task type 1. (b) Task type 2. (c) Task type 3. (d) Task type 4. (e) Task type 5.

one subtask can be executed in parallel with two other serial subtasks. For these five types of tasks, five groups of tasks with the same amount of computing tasks but different task topologies are selected, and the execution results of these five groups of tasks are compared separately to reflect the execution effects of different methods on tasks with different topologies.

Figure 10 shows the comparison of the average task delays obtained after three methods are used to compute and offload a set of tasks with several 20 different directed acyclic graphs in the same network environment. From the figure, it can be clearly seen that the task execution effect of task type 3 is significantly better than other task types, and the task with the highest task execution delay is the task of task type 1. For the task of task category 1, the 5 subtasks can only perform serial work, so the execution delay is the highest. In task type 3, up to 3 subtasks can be processed in parallel at the same time. By migrating the parallel processing tasks to different edge computing servers, the computing tasks of the three subtasks can be processed at the same time. It can be seen from the experimental results that the higher the degree of parallelism of the task, the more obvious the optimization effect after computing offloading, and the lower the delay obtained.

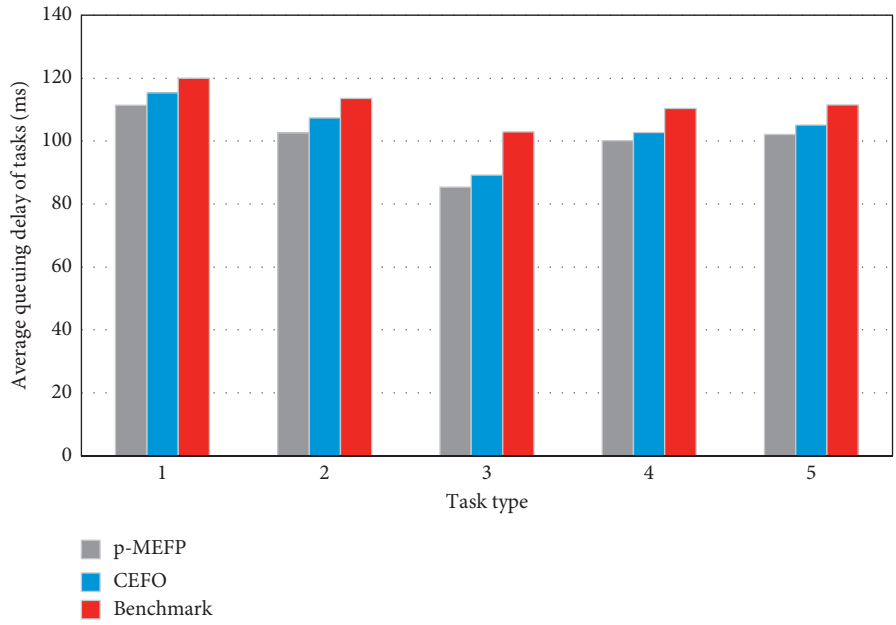

Figure 10: The average delay of computing offloading for different task types.

6. Conclusion

In this paper, the delay and energy consumption of edge computing offloading in the 5G network are analyzed firstly, according to which the goal of minimizing task delay has been proposed. A delay-aware offloading strategy, reducing the overall completion time of IoT applications by decomposing a computing task into several subtasks, is proposed which is expanded for multiuser situations. At the same time, the algorithm has been optimized for possible resource contention. To verify the performance of the proposed method, simulation experiments have been carried out. The results have shown that compared with the existing work, the proposed work can effectively reduce the overall task delay.

Data Availability

The basic data included in this study are provided in the supplementary information files.

Conflicts of Interest

The authors declare no conflicts of interest.

Supplementary Materials

The task requests data used in this study are stored in six files in the supplementary materials, which have the same format, and the number ranges from 50 to 300, respectively. In detail, the first to fifth columns in each piece of data are the task id, workflow id, start time, end time, and path length in sequence. (*Supplementary Materials*)

References

[1] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1133–1146, 2020.

[2] C. Shu, Z. Zhao, Y. Han, and M. Geyong, "Multi-user offloading for edge computing networks: a dependency-aware and latency-optimal approach," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 1678–1689, 2019.

[3] Z. Chang, L. Liu, X. Guo, and S. Quan, "Dynamic resource allocation and computation offloading for IoT fog computing system," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 3348–3357, 2020.

[4] A. Samanta and Z. Chang, "Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3864–3872, 2019.

[5] J. Zheng, Y. Cai, Y. Wu, and X. S. Shen, "Dynamic computation offloading for mobile cloud computing: a stochastic game-theoretic approach," *IEEE Transactions on Mobile Computing*, vol. 18, no. 4, pp. 771–786, 2018.

[6] X. Xu, Z. Fang, J. Zhang et al., "Edge content caching with deep spatiotemporal residual network for IoV in smart city," *ACM Transactions on Sensor Networks*, vol. 17, no. 3, pp. 1–33, 2021.

[7] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[8] Z. Ning, K. Zhang, X. Wang et al., "Intelligent edge computing in internet of vehicles: a joint computation offloading and caching solution," *IEEE Transactions on Intelligent Transportation Systems*, vol. 99, 2020.

[9] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: new paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[10] Z. Ning, P. Dong, X. Wang et al., "Mobile edge computing enabled 5G health monitoring for Internet of medical things: a decentralized game theoretic approach," *IEEE Journal on*

Selected Areas in Communications, vol. 39, no. 2, pp. 463–478, 2020.

[11] H. Yang, Y. Liang, J. Yuan, Q. Yao, A. Yu, and J. Zhang, "Distributed blockchain-based trusted multidomain collaboration for mobile edge computing in 5G and beyond," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 11, pp. 7094–7104, 2020.

[12] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Toward edge intelligence: multiaccess edge computing for 5G and internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6722–6747, 2020.

[13] Y. Zhai, T. Bao, L. Zhu, M. Shen, X. Du, and M. Guizani, "Toward reinforcement-learning-based service deployment of 5G mobile edge computing with request-aware scheduling," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 84–91, 2020.

[14] X. Xu, Q. Huang, Y. Zhang, S. Li, L. Qi, and W. Dou, "An LSH-based offloading method for IoMT services in integrated cloud-edge environment," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 16, no. 3s, pp. 1–19, 2021.

[15] Y. Liu, S. Wang, Q. Zhao et al., "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4961–4971, 2020.

[16] L. Chen, J. Wu, J. Zhang, H. N. Dai, X. Long, and M. Yao, "Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation," *IEEE Transactions on Cloud Computing*, p. 1. In press, 2020.

[17] M. Wang, T. Ma, T. Wu, C. Chang, F. Yang, and H. Wang, "Dependency-aware dynamic task scheduling in mobile-edge computing," in *Proceedings of 2020 16th international conference on mobility, sensing and networking (MSN)*, pp. 785–790, IEEE, Tokyo, Japan, December 2020.

[18] X. Li, D. Li, J. Wan, C. Liu, and M. Imran, "Adaptive transmission optimization in SDN-based industrial internet of things with edge computing," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1351–1360, 2018.

[19] 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*, vol. 21, no. 1, pp. 1–17, 2021.

[20] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: a survey, use cases, and future directions," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.

[21] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, "Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 235–250, 2019.

[22] X. Liu, J. Yu, J. Wang, and Y. Gao, "Resource allocation with edge computing in IoT networks via machine learning," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3415–3426, 2020.

[23] B. Shen, X. Xu, L. Qi, X. Zhang, and G. Srivastava, "Dynamic server placement in edge computing toward internet of vehicles," *Computer Communications*, vol. 178, pp. 114–123, 2021.

[24] P. Zhou, K. Shen, N. Kumar, Y. Zhang, M. M. Hassan, and K. Hwang, "Communication-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Internet of Things Journal*, vol. 99, p. 1, 2020.

[25] R. S. Pereira, D. D. Lieira, M. A. C. D. Silva et al., "RELIABLE: resource allocation mechanism for 5G network using mobile edge computing," *Sensors*, vol. 20, no. 19, p. 5449, 2020.

[26] Y. Jararweh, "Enabling efficient and secure energy cloud using edge computing and 5G," *Journal of Parallel and Distributed Computing*, vol. 145, pp. 42–49, 2020.

[27] M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao, and X. Shen, "Energy-efficient UAV-assisted mobile edge computing: resource allocation and trajectory optimization," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3424–3438, 2020.

[28] M. Merluzzi, P. D. Lorenzo, S. Barbarossa, and V. Frascolla, "Dynamic computation offloading in multi-access edge computing via ultra-reliable and low-latency communications," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 6, pp. 342–356, 2020.

[29] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-UAV-Enabled load-balance mobile-edge computing for IoT networks," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6898–6908, 2020.

[30] T. Cao, C. Xu, J. Du et al., "Reliable and efficient multimedia service optimization for edge computing-based 5G networks: game theoretic approaches," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1610–1625, 2020.

[31] S. Yang, "A joint optimization scheme for task offloading and resource allocation based on edge computing in 5G communication networks," *Computer Communications*, vol. 160, pp. 759–768, 2020.

[32] Z. Zhu, G. Han, G. Jia, and L. Shu, "Modified DenseNet for automatic fabric defect detection with edge computing for minimizing latency," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9623–9636, 2020.

[33] H. Tian, X. Xu, T. Lin et al., "DIMA: distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning," *World Wide Web*, pp. 1–24, 2021.

[34] X. Xia, F. Chen, Q. He et al., "Data, user and power allocations for caching in multi-access edge computing," *IEEE Transactions on Parallel and Distributed Systems*, p. 1, 2021.

[35] D. Harris, J. Naor, and D. Raz, "Latency aware placement in multi-access edge computing," in *Proceedings of 2018 4th IEEE conference on network softwarization and workshops (NetSoft)*, pp. 132–140, IEEE, Montreal, Canada, June 2018.

[36] V. D. Nguyen, T. T. Khanh, T. Z. Oo, N. H. Tran, E. N. Huh, and C. S. Hong, "Latency minimization in a fuzzy-based mobile edge orchestrator for IoT applications," *IEEE Communications Letters*, vol. 25, no. 1, pp. 84–88, 2020.

[37] Y. Han, Z. Zhao, J. Mo, C. Shu, and G. Min, "Efficient task offloading with dependency guarantees in ultra-dense edge networks," in *Proceedings of 2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, Waikoloa, HI, USA, December 2019.

[38] C. Shu, Z. Zhao, Y. Han, and G. Min, "Dependency-aware and latency-optimal computation offloading for multi-user edge computing networks," in *Proceedings of 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 1–9, IEEE, Boston, MA, USA, June 2019.