Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 3700135, 16 pages
https://doi.org/10.1155/2022/3700135

Research Article

An Effective Task Offloading Method for Separable Complex Mobile Terminal Tasks

Zemin Liu,[1] Na Zhou,[1,2] Yan Wang ,[1,2,3] Jian-Tao Zhou,[1,2,3] Haotian Zhang,[1] and Gang Xu[1]

[1]*Inner Mongolia Engineering Lab of Cloud Computing and Service Software, College of Computer Science, Inner Mongolia University, Hohhot, China*
[2]*Ecological Big Data Engineering Research Center of the Ministry of Education, Hohhot, China*
[3]*National and Local Joint Engineering Research Center of Mongolian Intelligent Information Processing, Hohhot, China*

Correspondence should be addressed to Yan Wang; 55234043@qq.com

Received 25 October 2021; Revised 9 January 2022; Accepted 21 January 2022; Published 14 February 2022

Academic Editor: Ting Bi

Due to limited energy and computing power of IoT devices, they cannot handle complex tasks. Edge computing technology effectively solves the requirements of computing power and response delay for complex tasks in devices by migrating computing power to the vicinity of IoT devices. For a separable complex task on IoT terminal, we focus on the effects of data distribution, dependencies, and offloading sequence of subtasks on its total delay when it is offloaded to edge servers. Through comprehensively considering these factors, we study the slicing and choreographing method during the offloading process of a complex task. Firstly, a task slicing method based on hierarchical clustering is presented and an improved hierarchical clustering algorithm is used to obtain the optimal solution of task partitioning. Secondly, a task choreographing method based on overlapping the longest path is presented. Finally, through the simulation experiments of complex tasks with different structures and loads, the effectiveness of our method is verified.

1. Introduction

In recent years, as the mobile Internet industry matures, the rapid explosion of the Internet of Things (IoT) leads the vigorous development of mobile intelligent terminal devices, which are widely used in transportation, health, entertainment, and other fields. At the same time, the applications deployed in IoT devices are becoming more and more complex. For example, they need to deal with large amounts of IoT data and complex processing processes. Limited by their own processing capacity and battery capacity, IoT devices have been unable to meet the needs of these tasks. Cloud computing emerges as a computing mode with unlimited supply of resources and becomes an effective supplement to terminal processing capacity. Mobile devices transfer data to remote terminals, use the resources of cloud data centers to complete efficient operations, and return the final results to users, so as to achieve the goal of fast data exchange. Compared with performing the task directly on the user terminal, the task processing mode of transferring data to the cloud is faster and more efficient, which can support mobile applications to achieve richer functions. However, such mode also has some disadvantages. For example, as users need to transfer amount of data to the cloud center for processing, the data transfer time is too long to exceed the effective delay requirement of the application. In addition, the link distance between mobile devices and cloud center is long, which is prone to interruption or instability, leading to the failure of the feedback results.

The appearance and application of edge computing solve the above problems to a certain extent. Edge computing provides cloud services and IT environment for IoT devices through sinking the processing capacity of the cloud platform to the network edge closer to IoT devices, which makes tasks on IoT devices be offloaded and processed more quickly. Compared with cloud computing, edge computing enables mobile devices to have a shorter data transmission path for offloading tasks, thus reducing the feedback and

transmission delay of data and results, meeting the needs of delay-sensitive tasks on IoT devices. Meanwhile, processing tasks at the edge is also helpful to relieve the traffic pressure on the network.

There are some drawbacks to offloading tasks to edge servers. Due to the limitation of edge server's own processing capacity, the processing time of offloaded tasks will increase when the computing requirement is relatively large. Such processing delay cannot also satisfy a delay-sensitive task, and some complex tasks even exceed the processing capacity of a single edge service. For this, there are two solutions. One way is to offload tasks to multiple edge servers through distributed computing to shorten the time delay of task execution. The other way is to combine the edge computing with the cloud computing to work together to complete the tasks, which can improve the processing capacity and shorten the transmission time. In both cases, we need to divide complex tasks into smaller ones and choreograph them to multiple edge servers or the cloud.

In this paper, we focus on the slicing and choreographing method of a complex delay-sensitive task at edge servers. The main research work and contributions include the two following aspects: (1) Aiming at the problem that the response time of offloading task is too long to meet the delay requirements of IoT devices, the task slicing method based on hierarchical clustering is improved to reduce the communication cost of subtasks on different servers and minimize the time consumption of task workflow while supporting the parallel and distributed execution of subtasks. (2) On the basis of task slicing, a subtasks choreography method combining static, dynamic, and the earliest start time of the task workflow is proposed, and a scheduling algorithm of task workflow on edge servers is designed.

2. Related Works

Mobile edge computing provides a promising solution for mobile terminal devices with limited computing capacity to complete complex, intensive, and sensitive computing tasks. Therefore, in the MEC system, it is very important to optimize the assignment and scheduling of mobile terminal tasks. In recent years, many strategies, methods, and algorithms have been proposed. In this section, we make a detailed introduction and analysis of representative work related to this paper.

Mobile edge computing provides enhanced computing power for mobile devices by deploying edge servers next to the communication base stations. When a mobile terminal receives a task request, the first issue to be decided is whether to offload part or all of the task to an MEC server. Considering the computing needs of different tasks on mobile devices, the optimization method of task partition ratios was proposed to minimize the maximum task latency. In [1], the authors divided multiple parts into a single subtask with prior knowledge and modeled the ordinal number relation of parts to guide the segmentation process in a circular way. In [2], each user could partition their computation task into offloading computing and locally computing parts in multiuser MEC networks. In [3], the offloading location of the task was further extended to the cloud server. Each task on the mobile device can be decided to be processed locally at its mobile device or offloaded to one of the edge servers or a cloud server. When the offloaded task is complex and the computing power of a single edge server is limited, an offload strategy is proposed to divide a task into some subtasks and deploy them in multiple servers. In [4], the authors assumed that each user's tasks were separable and proposed a distributed algorithm to obtain the hierarchical multilevel offloading decisions. In [5], a multiserver system with dynamic speed and power management was modeled as queueing systems, and then the issue of the optimal task dispatching on multiple heterogeneous server systems was addressed. In [6], the authors allocated computing tasks to suitable cores of mobile devices or the cloud in the MCC and proposed an optimization framework to minimize the total energy consumption and maximize the system reliability. For the task offloading problem of a heterogeneous multilayer MEC (HetMEC), the authors designed the latency minimization algorithm by jointly coordinating the resources among the end devices, multilayer MEC servers, and the cloud center in [7]. In [8], the authors designed LL-MLS algorithm to find an optimal partition of a given workload through task scheduling and energy allocation strategies. In [9], the authors proposed an energy-aware cooperative routing (ECoR) scheme for optimal handling of task offloading between source and target UAVs in a gridlocked swarm.

Offloading the task to the edge computing system not only provides the task with expanded processing capacity but also brings with it the transmission delay caused by the offloading process. Therefore, the allocation of wireless resources for offloading task is also the focus of many research works. In [10], the authors transformed the problem of joint task assignment and wireless resource assignment into a mixed-integer nonlinear program (MINLP) and proposed a suboptimal solution algorithm based on relaxation convex problem to reduce time delay for offloading tasks. In [11], the problem of task assignment in the MEC in the return network was solved by a similar method. In [12], an online adaptive task allocation and computing offload strategy was proposed, which coordinated and optimized the wireless and computing resource allocation by considering dynamic wireless conditions and service delay constraints.

In order to efficiently implement the allocation and deployment of multiple tasks in the MEC, some researchers regard the problem as a joint optimization problem considering various offloading conditions. In [13], the authors minimized energy consumption for all devices and their task delay constraints by cooptimizing communication and computing resource allocation on devices and mobile edge servers. Considering the task completion time and the mobile device energy consumption, the authors in [14] proposed a heuristic offloading decision algorithm (HODA), which jointly optimized the offloading decisions, communication, and computing resources to maximize the system utility. In order to reduce the complexity of the joint optimization problem, the original problem was decomposed into two subproblems in [15–21]. In [15], the authors

addressed the resource allocation problem using the convex and quasi-convex optimization techniques and solved the problem of task assignment by a heuristic algorithm. In [16], the task partitioning subproblem was taken as a set of univariate optimization problems, which can be easily solved, and the task scheduling subproblem was solved through a heuristic algorithm. In [17], the problem of resource allocation was further decomposed into two stages: the computing resource optimization and the communication resource allocation. The authors proposed a subchannel allocation scheme, and then the transmission power allocation was considered as a convex optimization problem based on the scheme and was solved by the Lagrange multiplier method. For the resource allocation scheme, the authors in [18] proposed a computing framework based on the weighted sum of task completion time and energy consumption in the MEC system, while the authors in [19] proposed a task shunting and resource allocation algorithm based on Deep-Q network. In [20], the authors considered users' risk-seeking or loss-avoidance behaviors in their final decision. In [21], the authors proposed the energy-efficient multihop communication solution in smart city environment.

In addition to finding a better task offloading strategy by optimizing the allocation and consumption of communication resources and computing resources during task offloading, some other factors, such as shared data among tasks, offloading sequence of tasks, and the mobility of mobile devices, also have an important impact on the efficiency of task offloading. In [22], the authors studied the task assignment algorithm in data shared mobile edge computing systems and proposed three algorithms to deal with holistic tasks and divisible tasks, respectively. In [23], an adaptive slicing method for decentralized workflow based on clustering was proposed. Then a data-related task scheduling algorithm based on the correlation task model was designed, which gave an evaluation function to reduce the intercore communication during the process of task execution by assigning highly the correlated tasks to the same core. In [24], the authors gave full consideration to the mobility of user in the MEC and then proposed a device-to-device (D2D) cooperation method to expedite the task execution of mobile user by leveraging proximity-aware task offloading. In [25], the user mobility and network constraints were considered, and a lightweight heuristic solution was proposed for fast scheduling. In [26], a task allocation solution for optimizing latency and service quality was proposed to support the mobility of vehicles, in which the constraints on service latency, quality loss, fog capacity, stationary task allocation, and mobile fog nodes were taken into account. In [27], the effective task offloading scheme in the MEC was designed, in which the tasks are offloaded to the adjacent servers at the next AP in the direction of vehicle driving. In [28], the authors emphasized the importance of optimizing operation sequence in multiuser MEC system and established a computation offloading model to optimize the task operation sequences and starting times for uploading, executing and downloading, and duration times for uploading and downloading. In [29], a spatiotemporal framework based on stochastic geometry and continuous time Markov chains was proposed. The experimental results showed that the framework can find the optimal number of edge servers for parallel computing of the user task. In [30], the authors studied the scheduling method of parallel tasks merging and scheduling for parallel deep learning applications in the MEC.

When the tasks of mobile terminals are offloaded to the edge computing system, particularly the tasks that can be divided are offloaded to different edge servers, and the execution sequence of tasks on edge servers is the key to determine the actual execution efficiency of tasks. For this, the authors in [31] focused on the problem of providing QoS and performance guarantees to divisible loads and then proposed a linear algorithm for real-time divisible load scheduling by eliminating the need to generate exact schedules in the admission controller. In [32], the authors adopted a Markov decision process to handle the problem of computation task scheduling for MEC systems, where the computation tasks were scheduled based on the queueing state of the task buffer, the execution state of the local processing unit, and the state of the transmission unit. In [33], a partitioned fixed-priority real-time scheduling based on dependent tasks split on homogeneous multicore platform was proposed, which converted dependent tasks into a series of sequential jobs and obtained the interrelated subtasks path as well as synthetic deadlines through the B-tree task model. In [34], the authors creatively proposed a deep learning architecture based on tightly connected network and proposed a corresponding multitask parallel scheduling algorithm. In [35], a peer-to-peer (P2P) enhanced task scheduling framework to minimize the average task duration in device-to-device (D2D) network was proposed. In the framework, an iterative algorithm based on alternating optimization and sorting technology was used to solve the approximate optimal scheduling solution.

To sum up, when a complex task is divided and offloaded to multiple edge servers or cloud, the efficiency of task offloading is affected by many factors. The above studies put forward some effective task offloading strategies from the perspectives of processing capacity of mobile terminals and servers, communication channel allocation in the process of task offloading, and optimization of multitask deployment in multiple servers. However, these studies pay little attention to the effect of data interaction between subtasks and the offloading sequence of subtasks on the execution delay of the whole task after tasks are divided into subtasks. Obviously, these factors also have a great impact on the efficiency of subsequent task scheduling. Although papers [31–35] focus on these two factors to optimize the task offloading process, they are not considered as a whole. However, task slicing is closely related to its offloading sequence, and different slicing schemes should correspond to different offloading sequence to optimize the execution delay of the task to the maximum extent. So we focus on the two following problems in the offloading process: (1) in the distributed deployment of complex tasks on edge servers, the data dependencies among the subtasks are fully considered to minimize the communication delay caused by such data

dependencies during task execution. (2) In the process of task scheduling, the effect of subtask offloading sequence on task execution is considered, and the execution delay of the whole task is minimized by parallel subtask offloading and subtask execution.

3. Problem Definition and Formalization

Here, we consider a mobile task offloading scenario with multiuser, multiedge servers. Users' mobile devices can connect and communicate with base stations covering their signals. Edge servers are uniformly deployed near these base stations. At least one edge server is deployed near each base station. Assume that there are m mobile terminal devices and n edge devices in this scenario, $U = \{u_1, u_2, \ldots, u_m\}$ represents the set of mobile terminal devices, and $S = \{s_1, s_2, \ldots, s_n\}$ represents the set of edge servers. The service request sent by each terminal device can be divided into a series of subtasks. We use a directed acyclic graph (DAG) to represent the tasks offloaded by the mobile terminal and the relationships among them, represented as $G(T, E, C(T), W(E))$, where $T = \{t_1, t_2, \ldots, t_k\}$ represents all subtasks offloaded by the mobile terminal and k is the number of subtasks; $E = \{e_{ij} | \text{ if the output of subtask } t_i \text{ is an input to subtasks } t_j\}$ represents data dependencies between subtasks; $C(T) = \{c_1, c_2, \ldots, c_k\}$ represents the workload of each subtask in set T; that is, c_i is the CPU cycle of subtasks t_i; $W(E) = \{w_{ij} | \text{ if } \exists e_{ij},\ W(e_{ij}) = w_{ij}\}$ represents the size of the input data from t_i to t_j. An example of a mobile terminal task offloaded on edge servers is shown in Figure 1.

Given $G(T, E, C(T), W(E))$, we need to offload multiple subtasks with dependent relationships to the edge service system. If all subtasks in T are deployed on the same edge server, the computing capacity constraints of a single server and the serial execution of tasks may lead to too long feedback delay of the task to meet its needs. In order to take advantage of edge service system to better meet the demands of mobile terminal, we need to offload the task to multiple edge servers, respectively, make some tasks in parallel execution, and shorten the overall delay of the task. For example, the subtasks t_2, t_3, and t_4 can be executed in parallel in Figure 1. A new challenge is that the data dependencies between subtasks introduce new transmission delays. In particular, when the subtasks with large data dependencies are deployed on different servers, the new latency introduced may even outweigh the time savings in the process of executing the tasks in parallel. To solve this problem, the goal of this paper is to first find a task partitioning scheme based on the dependencies between subtasks; we call it task slicing, which can reduce the introduction of new delay as much as possible while deploying all subtasks in a distributed way. In addition to the task slicing affecting the execution delay of the task, the offloading order of the subtasks also has a certain impact on the feedback delay of the task. However, most of the existing studies mainly ignore this problem. In fact, due to the limited wireless communication resources in the MEC environment, when multiple tasks are offloaded at the same time, each task will receive less wireless resources, which will inevitably increase the transmission delay of offloaded subtasks. In $G(T, E, C(T), W(E))$, the subtasks do not need to be executed at the same time, so it is not necessary to simultaneously offload subtasks to different edge server. We just need to make sure that a subtask is offloaded before it is executed, which can maximize offloading bandwidth allocation of the subtasks so as to shorten their transmission delay. Therefore, we will study the task choreography method based on task slicing to optimize the overall delay of the task. The problem in this paper is formally described as follows:

$$\min_{G} \text{ total Time} = T_{\text{exec}}^{\text{end}}(\text{Max}T(\text{Ord}(\mathbb{S})) - T_{\text{offload}}^{\text{start}}(\text{Min}T(\text{Ord}(\mathbb{S})), \tag{1}$$

where

$$\mathbb{S} = \{TL_1, TL_2, \ldots, TL_{k'}\}, \quad (k' \le k), \tag{2}$$

$$\text{Ord}(\mathbb{S}) = \{\langle TL_i, pr_i\rangle |, \quad i \in [1, k'],\ pr_i \in R,\ pr_i > 0\}, \tag{3}$$

s.t.

$$TL_i \subseteq T, \quad (i \in [1, k']), \tag{4}$$

$$\bigcup_{i \in [1,k']} TL_i = T \text{ and } \bigcap_{i \in [1,k']} TL_i = \Phi, \tag{5}$$

$$\text{T}_{\text{exec}}^{\text{start}}(t_i) \ge \text{Max}\left(\text{T}_{\text{exec}}^{\text{end}}(t_j)\right), \quad (i, j \in [1, k]) \text{ if } t_j \in \text{PRE}(t_i) \text{ and } t_i,\ t_j \in TL_p, \tag{6}$$

$$\text{T}_{\text{exec}}^{\text{start}}(TL_i) \ge \text{Max}\left(\text{T}_{\text{exec}}^{\text{end}}(TL_j) + \text{T}_{\text{tran}}(TL_j, TL_i)\right), \quad \left(i, j \in [1, k'] \text{ and } TL_j \in \text{PRE}(TL_i)\right), \tag{7}$$

$$\text{T}_{\text{offload}}^{\text{end}}(TL_i) \le \text{T}_{\text{exec}}^{\text{start}}(TL_i), \quad (i \in [1, k']), \tag{8}$$

$$T_{\text{offload}}^{\text{end}}(TL_i) \ge \sum_{t_j \in TL_i} T_{\text{offload}}^{\text{end}}(t_j) - T_{\text{offload}}^{\text{start}}(t_j). \tag{9}$$

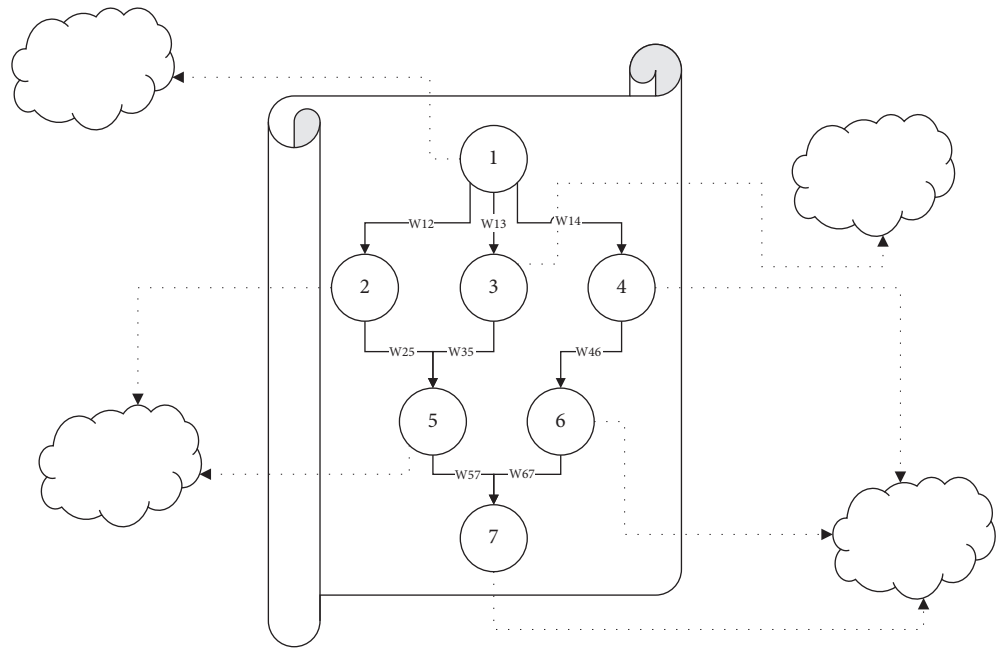


Figure 1: An example of complex task offloading.

Equation (1) is our optimization goal to find a slicing and choreography scheme for task offloading of mobile devices, that is, $\mathbb{S}$ and Ord($\mathbb{S}$), which minimizes the response time of the overall task. As shown in equation (2), $\mathbb{S}$ is a partition of T. Each task slicing corresponds to a priority, which represents the order in which subtasks are offloaded, as shown in equation (3). $T^{start}_{exec}(t)$ and $T^{end}_{exec}(t)$ represent the start and end times for executing t, respectively. $T^{start}_{offload}(t)$ and $T^{end}_{offload}(t)$ represent the start and end times for offloading t, respectively. Min$T$ and Max$T$ are used to obtain the last and first task slices in the choreography scheme, respectively. Equations (4)–(9) represent the constraint conditions that need to be satisfied when offloading the task, where $T_{tran}(TL_j, TL_i)$ represents the data transmission time between TL_j and TL_i and $PRE(TL_i)$ is used to obtain the preorder task slicing of TL_i. Equations (4) and (5) represent the basic requirements of task slicing. Equation (6) indicates that if there is a dependency relationship between the subtasks in a task slicing, the start time of the subsequent subtasks must be later than the end time of all the preordering subtasks. Equation (7) indicates that if there is a dependency relationship between the subtasks in different task subsets, the subsequent subtask cannot be executed until it has received the output data of all the preordering subtasks to the server where it was deployed. Equation (8) indicates that any task slicing must be offloaded to the corresponding edge server before it begins to execute. Equation (9) indicates that when multiple subtasks need to be offloaded on the same edge server, they must be offloaded in sequence.

4. Task Slicing and Choreographing Model

For complex mobile terminal requests that can be divided into multiple subtasks, we can distribute these tasks on multiple edge servers to improve the processing capacity, which is conducive to reducing the execution delay of terminal tasks. In this section, we will establish a slicing and choreographing model for complex tasks to minimize the transmission and computation delay in the process of task offloading. Here, we name our method SSCS (Slicing Similarity and Choreograph Sequence).

4.1. Task Slicing Method Based on Data Dependencies. The objective of task slicing model is to optimize the parallel execution of subtasks on different edge servers and minimize the delay caused by data transmission between subtasks. We propose a slicing model based on task workflow, in which the concept of task similarity is defined based on the dependency relationship between subtasks. For two subtasks, if there is a large amount of data exchange between them but less contact with other subtasks, they will be divided into the same cluster. The related definitions are given below.

Definition 1. Subtask (t). It istThe basic unit of a task slice, which refers to a task that cannot be divided again, corresponding to an element in the task set T.

Definition 2. Task slice (TL). It is the basic unit of task deployment. After a task is split into multiple subtasks, the similar subtasks will be grouped into the same subtask set, called task slice.

Definition 3. Task hierarchy association matrix (A). It is a hierarchical logic relation of task execution for given $G(T, E, DS(T), W(E))$, represented as $A_{(r*k)}$, where k is the number of subtasks in T and r is the number of logical levels that the task workflow needs to be executed at least, formally

$$a_{ij} = \begin{cases} 1, & \text{if } \lambda(t_j) = \text{NULL at } L(t_j) = i, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

$L(t_j) = i$ means t_j is at the i-th layer. $\lambda(t_j) = \text{NULL}$ means that all the preceding subtasks of t_j have been executed, and task t_j can start to execute.

Definition 4. Direct correlation of subtasks (dirCorr). In G, if $e_{ij} \in E$ exists, then the direct correlation degree between task t_i and t_j is represented by

$$\text{dirCorr}_{ij} = w_{ij}. \tag{11}$$

It represents the amount of direct communication data between two subtasks, and the larger the value is, the greater the probability that the two subtasks should be deployed to the same edge server.

Definition 5. Dependency between subtasks (D). In G, the dependencies between subtasks can be divided into three categories: single dependency (D^{single}), split dependency (D^{split}), and joint dependency (D^{join}). D_{ij}^{single} means that task t_j depends entirely on task t_i; that is, $\forall x$, if $x \neq j$, then $e_{ix} \notin E$, and, $\forall y$, if $y \neq i$, then $e_{yj} \notin E$. D_{ij}^{split} means that task t_j depends partly on task t_i, that is, $\exists x$; make $e_{xj} \in E$ and $x \neq j$. D_{ij}^{join} means that a part of the input of task t_j depends on task t_i, that is, $\exists x$; make $e_{xj} \in E$ and $x \neq i$. If $D_{ij} = \text{false}$, then there is no direct dependency between t_i and t_j.

Definition 6. Subtask dependency correlation (depCorr). Based on different types of dependency relationships between subtasks, the dependency correlation degree between t_i and t_j is represented by

$$\text{depCorr}_{ij} = \begin{cases} 1, & \text{if } D_{ij}^{\text{single}} = \text{true}, \\ \dfrac{w_{ij}}{\sum_{i=1}^{k} w_{ix}}, & \text{if } D_{ij}^{\text{split}} = \text{true}, \\ \dfrac{w_{ij}}{\sum_{i=1}^{k} w_{xj}}, & \text{if } D_{ij}^{\text{join}} = \text{true}, \\ 0, & \text{if } D = \text{false}, \end{cases} \tag{12}$$

where wij represents the amount of communication data between t_i and t_j; if there is no dependency relationship between them, then $w_{ij} = 0$.

Definition 7. Subtask distribution correlation (disCorr). It reflects the influence of data distribution between subtasks on task partition results. The larger the data traffic between subtasks, the greater the probability of the subtasks coupling into the same slice. The communication correlation degree between t_i and t_j can be expressed by

$$\text{disCorr}_{ij} = \frac{\left(Corr_{ij}^{\text{in}} + Corr_{ij}^{\text{out}}\right)}{2}, \quad (i, j \in [1, k] \text{ and } i \neq j). \tag{13}$$

From the above equation,

$$\text{Corr}_{ij}^{\text{in}} = \frac{\sum_{x=1}^{k}\left(w(e_{xi}) + w(e_{xj})\right)}{2}, \tag{14}$$

$$\text{Corr}_{ij}^{\text{out}} = \frac{\sum_{x=1}^{k}\left(w(e_{ix}) + w(e_{jx})\right)}{2}. \tag{15}$$

In equation (13), $\text{Corr}_{ij}^{\text{in}}$ and$\text{Corr}_{ij}^{\text{out}}$ are the correlation degrees calculated by the input and output data traffic of t_i and t_j, respectively. $w(e_{ij})$ represents the communication data volume from t_i to t_j. If there is no dependency between them, then $w(e_{ij}) = 0$.

Definition 8. Subtask computation time (exetime). Due to the different sizes of subtasks, their computation time will be different. Assume that the computing power of the CPU is C^{Edge}; $exetime_i$ can be expressed as follows:

$$\text{exetime}_i = \sum \frac{ps_i * Mpc}{C^{\text{Edge}}}, \tag{16}$$

where Mpc represents the CPU cycles required to process a unit of data.

Definition 9. Sequential execution time between subtasks (exetime_{ij}). If there is a sequential dependency between two consecutive subtasks, their sequential execution time is the sum of their respective execution times; otherwise, the value is 0.

$$\text{exetime}_{ij} = \begin{cases} \text{exetime}_i + \text{exetime}_j, & \text{if } D = \text{true}, \\ 0, & \text{if } D = \text{false}. \end{cases} \tag{17}$$

Definition 10. Similarity between subtasks ($T\ D$). It represents the degree of comprehensive correlation between subtasks. The similarity between t_i and t_j is represented by

$$TD_{ij} = \frac{\left(\text{dirCorr}_{ij} + \text{exetime}_{ij}\right) \times \text{depCorr}_{ij} \times \text{dirCorr}_{ij}}{\left(\text{disCorr}_{ij} - \text{dirCorr}_{ij}\right)}. \tag{18}$$

Task similarity comprehensively measures the correlation between two subtasks in the whole workflow system from the direct data dependency between tasks and the relative importance of such dependency in the whole task flow, which serves as the basis for further coupling subtasks. First of all, dirCorr_{ij} is the main factor that determines whether the subtasks can be aggregated into a task slice. In addition, depCorr_{ij} reflects the degree of association between t_i or t_j and other tasks; the lower the degree of association between t_i or t_j and other tasks, the higher the probability that they are aggregated into a task slice. $exetime_{ij}$ represents

the computation time required to complete them. Dividing more subtasks that need to be executed sequentially into a task slice will help to avoid the delays caused by data transfer between subtasks. disCorr_{ij} reflects the degree of correlation between t_i and t_j and their precedence and postorder subtasks. The smaller the degree of correlation, the higher the probability that t_i and t_j are grouped into one task slice.

An example is given in Figure 2, where the weights between nodes represent the reciprocal of similarity between different subtasks. Firstly, starting from the first node of the task workflow, we look for the subtasks that can be grouped into one task slice from top to bottom. The rule of merging is that each subtask is merged with one of the subsequent subtask which has the lowest weight with it until the last node in the workflow. Secondly, starting from the last node of the task workflow, we continue to look for the subtasks that can be merged into one task slice from bottom to top. The rule of merging is that each task is merged with one of the preceding tasks which has the lowest weight with it until the first node in the workflow. For example, in Figure 2, nodes t_1, t_3, and t_7 are merged into one task slice, and nodes t_2, t_5, t_{10}, and t_{11} are merged into one task slice. Then, the minimum weight of the merged subtasks is set as the merge threshold. For those subtasks that are not merged, they are merged when the weight between continuous subtasks is less than the threshold. The threshold value is obtained by comprehensively considering all the similarity of the whole workflow. According to the rule, nodes t_4 and t_9 are merged into one task slice.

In the following, an improved horizontal clustering algorithm is given to solve the slicing scheme based on the similarity between subtasks, as shown in Algorithm 1.

Algorithm 1 provides a method to determine the optimal slicing scheme of task workflow under the premise of a given number of edge servers. The value of SM can be allocated statically according to the resource situation in the MEC system or solved dynamically by optimizing the overall computation delay of the task workflow. In Algorithm 1, lines 6–9 are used to calculate the similarity between the subtasks in T by using equations (10)–(18). Lines 10–18 are used to solve the task slicing scheme based on the improved hierarchical clustering process. Here, the concurrency of task slices is mainly considered, and subtasks at the same level cannot be divided into the same task slice, as shown in line 15.

4.2. Task Slice Choreography Method Based on the Longest Overlapping Path. Next, we need to choreograph these subtasks for offloading. Our goal is to accomplish sequential offloading of task workflow and shorten the task wait delay for executing while offloading. In this paper, we propose a subtask choreography method based on the longest overlapping path by analyzing the logical relationship and execution constraints among subtasks. The definitions are given below.

Definition 11. The computation time of task slice ($D^{\text{comp}}(TL)$). It refers to the sum of execution delays of all subtasks in the task slice.

$$D^{\text{comp}}(TL) = \sum_{t_i \in TL} \text{exetime}_i. \tag{19}$$

Each task slice must wait until all the subtasks on which it depends have been completed before it begins to execute. The earliest start time of a task slice is determined by the longest path from the initial task slice to the last task slice. Therefore, we first carry out static sorting for all tasks slices. Here, we give an example for the task slice hierarchical relationship, as shown in Figure 3.

The following is a brief description of static sorting rules. The node in the first layer is the first task slice of task workflow, its subsequent task slices are in the second layer, and so on. In general, when the task slice at layer i completes, the task slices at layer $i+1$ can start executing. But it is not strict. For example, when TL_3 is complete, TL_5 and TL_6 can be executed regardless of TL_2. If the computation delay of TL_5 or TL_6 is significantly higher than that of TL_4, then whether to offload TL_2 or TL_3 first will have an impact on the overall delay of the task workflow. Obviously, in this case, TL_3 should be preferred. In addition, TL_5 and TL_6, which are deployed on different edge servers, can be executed in parallel. But, compared to TL_6, TL_5 has more subsequent tasks. When the execution delay of TL_5 is not greater than that of TL_6, we should offload TL_5 first, so that it is executed more earlier than TL_6. The offloading order of task slices has an important effect on the computation time of the task slices on the edge servers. Here we define the earliest execution start time for a task slice.

Definition 12. The earliest start time of the task slice ($T^{\text{start}}_{\text{exec}}(TL)$). Each task slice must wait until all the subtasks on which it depends have been executed before it executes. The earliest start time of task slice TL is determined by the longest path from the initial task slice to this task slice, and the calculation formula is represented by

$$T^{\text{start}}_{\text{exec}}(TL_i) = \begin{cases} 0, & \text{if } L(TL_i) = 1, \\ \max\limits_{TL_x \in PRE(TL_i) \text{ and } L(TL_x) = L(TL_i) - 1} \left\{ \dfrac{T^{\text{start}}_{\text{exec}}(TL_x) + D^{\text{comp}}(TL_x) + w_{xi}}{B^{\text{StoS}}} \right\}, & \text{if } L(TL_i) > 1, \end{cases} \tag{20}$$

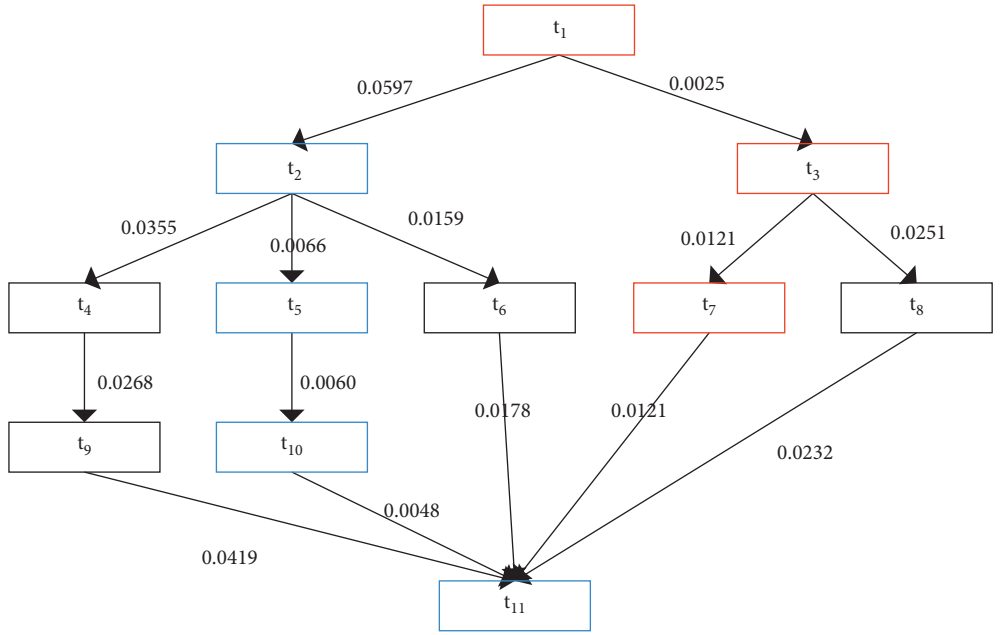


Figure 2: A task workflow example.

```
(1) function [S] = Slice (G, SM, A)
(2) Input: G
(3)       SM//Number of edge servers, 0 < M < m
(4)       A//Logical hierarchy matrix
(5) output: S
(6) InitNum(T)//Initializes the subtask
(7) taskNum = Count(T)//the number of T
(8) for i = 1: taskNum
(9)    TD = TaskSim (G)//TD is sliceNum * sliceNum matrix, and the similarity between sub-tasks is calculated
(10) sliceNum = taskNum
(11) while true
(12) if sliceNum ≤ SM
(13)       break;
(14) Stemp = MaxSim (TD);
(15) if Notlevel (Stemp, A)//Tasks at the same logical level cannot be divided into a task slice
(16)       Cluster = Merge (Stemp)//Task clustering, forming a new task slice division
(17)       sliceNum = Count (Cluster)
(18) S = Cluster
```

Algorithm 1: Task slicing algorithm based on similarity between subtasks.

where the longest path of task slice TL_i is equal to the longest path of all its presequence task slices, its own computation time, and the transmission delay between it and other task slices.

As shown in Figure 4, the subtasks are executed in the order of $TL_1, TL_3, TL_2, TL_5, TL_6, TL_4, TL_7, TL_8$ according to the static sort. However, since the size of each subtask is different, the computation time is also different. Scheduling tasks in a statically sorted manner can cause too much delay in the execution of the overall task workflow. Therefore, we need to combine static sorting with task slice's earliest start time to produce a comprehensive sorting result.

Next, we use the concept of task priority to represent the choreography scheme $\mathrm{Ord}(\mathbb{S})$ of task slice set $\mathbb{S}$. The priority of each task slice is determined by its latest offloading time. Let the priority of the last task slice in the task workflow be 0. The higher the priority of the task slice is, the earlier it should be offloaded. The related definitions are as follows.

Definition 13. Task slice priority ($\mathrm{Prio}(TL)$). It represents the time to offload the task, that is, the latest time for the task slice to be offloaded. The calculation formula of TL_i priority is as follows:

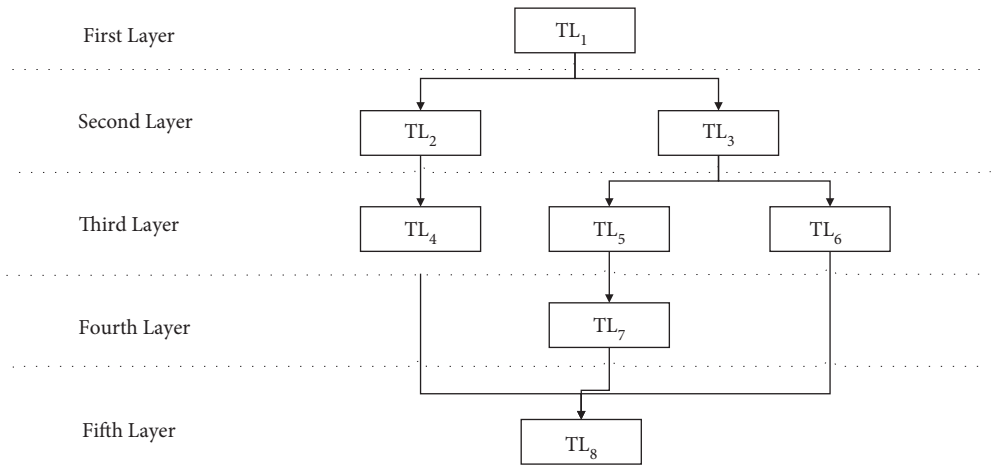


Figure 3: Task slice hierarchical association relationship.

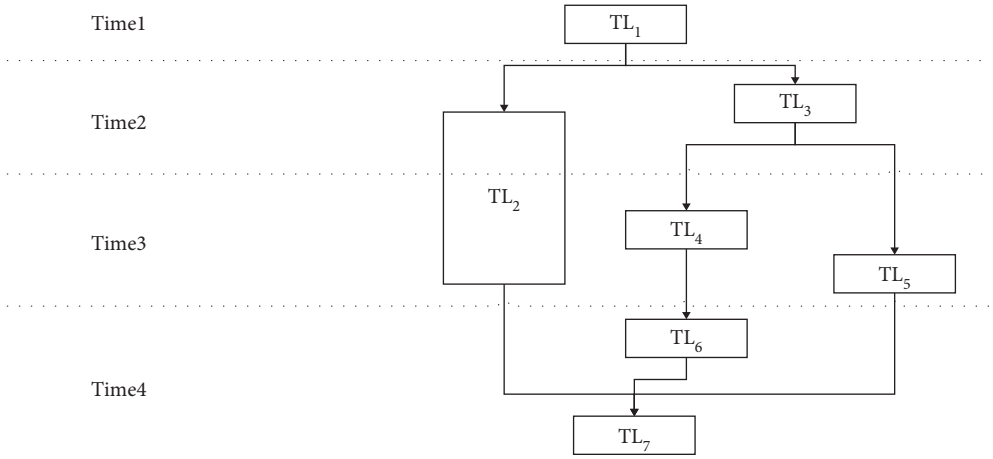


Figure 4: Diagram of the earliest start time of task slices.

$$\mathrm{Prio}\left(TL_i\right) = D^{\mathrm{offload}}\left(TL_i\right) + D^{\mathrm{comp}}\left(TL_i\right) + \max_{TL_x \in SUC\left(TL_i\right) \text{ and } L\left(TL_x\right)=L\left(TL_i\right)+1}\left(\frac{w_{ix}}{B^{\mathrm{StoS}}} + \mathrm{Prio}\left(TL_x\right)\right), \tag{21}$$

where $SUC(TL_i)$ represents the set of subsequent task slices of TL_i, which have direct data dependence between TL_x and TL_i, $\mathrm{Prio}(TL_x)$ is the task priority of TL_x, and B^{StoS} represents the channel bandwidth between edge servers.

pr_i in the optimal choreography $\mathrm{Ord}(\mathbb{S})$ is the optimal start time of the transmission corresponding to TL_i; in order to ensure that each task slice has been offloaded before it is executed on the edge server, $pr_i \le \mathrm{Prio}(TL_i)$. The task choreography method based on overlapping longest path can solve the best offload time for each task slice in T, so as to maximize the parallelization of the transmission and execution of subtasks, to realize the optimization goal in equation (1).

As shown in Figure 5, the choreography of subtasks is as follows: (1) Firstly, a static sort is done. For example, t_1 is executed in the first sequence, t_2 and t_3 are executed in the second sequence, and subtasks t_4, t_5, t_6, t_7, and t_8 are executed in the third sequence. (2) After that, all the subtasks are choreographed according to the earliest start time. It can be seen that the earliest start time of t_7 and t_8 is earlier than that of t_4, t_5, and t_6, so, in the third execution sequence, t_7 and t_8 should be executed earlier than t_4, t_5, and t_6. (3) Finally, the above two sequences are dynamically sorted according to the following rules: the nodes with more children take precedence or the nodes whose child nodes have high computation time take precedence. In Figure 5, t_4 and t_5 have more child nodes, and the computation delays of their child nodes are longer, so t_4 and t_5 have priority over t_6, t_7, and t_8.

We design a heuristic algorithm to solve the subtask choreography scheme (Algorithm 2).

The input of Algorithm 2 is the output of Algorithm 1. It uses dynamic iterative optimization to solve the optimal choreography scheme $\mathrm{Ord}(\mathbb{S})$, namely, the offloading

```
(1) function [S] = choreography (G, S, En)
(2) Input: G//Task workflow
(3)        S//Task slice scheme
(4)        En//Environmental parameters, including channel bandwidth, server processing capacity, etc
(5) output: Ord(S)
(6) CreTree(G, S)//Build the number of task slice levels
(7) Init(En)//Initialize the offload environment
(8) for i = 1:|S|
(9) T(i) = ExeTime(TL_i)//Calculate the earliest start time of the task slices
(10) KP = LogicP(S, T)//Find the longest path for the task slices to execute
(11) NKP = DelP(S, KP)//Get the task slices not in the longest path
(12) Cons = priority(S, ExeTime)//Obtain scheme constraint
(13) Sord = rand (Popsize, Cons, S)//Program population size
(14) while (k ≤ maxnum)
(15) for i = 1: Popsize
(16)        F(i) = fitness (G, Sord)//Set the optimization target of the heuristic algorithm
(17) [globlalMinT, ordi] = min(F)
(18) for i = 1: Popsize
(19)        Sord = IteV (Cons, KP, NKP)//Optimize the population of the choreography scheme under the constraints
(20) [BestMinT, bestord] = min (global, ordi)
(21) Ord(S) = BestMinT
```

Algorithm 2: Task choreography based on the longest overlapping path.

sequence and timing of each task slice in S. In this algorithm, lines 6–12 are used to calculate all kinds of time delays during the offloading, computation time of each task slice, and the computation time constraints of each task slice according to equations (19)–(21). In lines 13–21, a heuristic optimization algorithm is used to find the task choreographing scheme that minimizes the overall computation delay of the task workflow under the execution constraints. The combination of Algorithms 1 and 2 can achieve the optimal task slicing and choreographing scheme under the condition of a specific number of edge servers. In the case of sufficient resources of the MEC system, we can traverse from 1 to k (number of T subtasks) to find the optimal number of task slices, that is, the optimal number of edge servers for distributed deployment of the whole task workflow.

The computational complexity of an algorithm is determined by the number of basic operations when the input size is N. The SSCS algorithm proposed in this paper is a heuristic algorithm based on discrete optimization. The algorithm consists of generating the initial solutions, generating neighborhoods, judging the infeasible task scheduling list, and removing the infeasible task scheduling list. Since the generation of the initial solutions is constrained by the earliest start time of a task, the computational complexity of the operation is determined by the horizontal clustering result of the task workflow, that is, $O(N\log N)$. Similarly, the computational complexity of generating neighborhoods is determined by the maximum number of parallel tasks at each level, and the size does not exceed $O(C^2_{\log N})$. The computational complexity of judging the infeasible task scheduling list and removing the infeasible task scheduling list is $N^* L$, where L is the length of the infeasible task list. So, the overall complexity of the SSCS algorithm is $O(\text{Max\_Gen}^* N\log N)$, where Max_Gen is the maximum number of iterations.

5. Experiment and Analysis

In order to verify the effectiveness of the task slicing and choreographing method proposed in this paper, we set up two groups of simulation experiments. In the experiment, we first generated the complex task workflow of mobile devices covering the three data dependencies and assigned corresponding parameters to each subtask and MEC environment. The setting range of main parameters is shown in Table 1.

5.1. Verification Experiments of Task Slicing Method. Firstly, in order to prove the advantages of task slicing method, we use HPD (Hierarchical Process Decentralization) algorithm and HIPD (Hierarchical Intelligent Process Decentralization) in papers [36, 37] to generate subtask slicing and then conduct comparative experiments with our algorithm. The HPD algorithm applies breadth-first search/traversal algorithm to find the most relevant, closely related, and parallel activities in the workflow view and then encapsulates the closely related activities in the same broker to reduce the need for interbroker messaging. The HIPD algorithm combines HPD and a frequent path mining algorithm together. In this experiment, the task workflow to be offloaded by the mobile device and the data dependency relationship between subtasks are shown in Figure 5. We use these two methods to generate task slices of different granularity and compare the results with the slicing results of our algorithm. The partitioning results of the three slicing methods for the task workflow are shown in Table 2.

When the task is offloaded to the server, this group compares the advantages and disadvantages of the results of different task slices from three aspects: the overall computation delay of the task workflow, the load of the edge servers

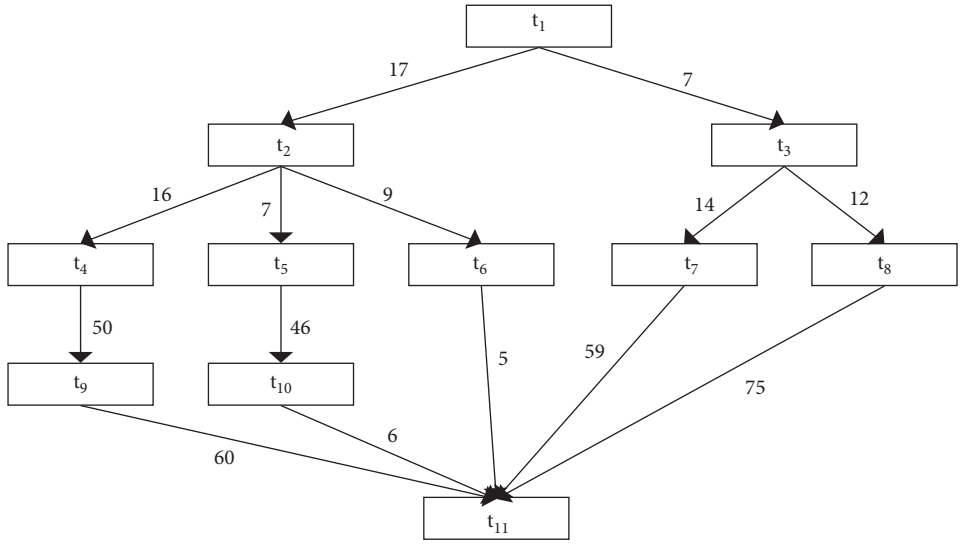


Figure 5: An example of dynamic choreography of tasks.

Table 1: Setting of main experimental parameters.

Parameter name	Parameter value range	Parameter meaning
ps_i $(i \in [1,k])$	1–100 M	The size of the task load
w_{ij} $(i, j \in [1,k])$	1–100 M	The amount of data transferred between tasks
Mpc	10^2 cpu cycles $\cdot$ M^{-1}	CPU cycles per M data to process
C^{Edge}	10^3 cpu cycles $\cdot$ s^{-1}	CPU cycles per second which the edge server can process
B^{StoS}, B^{DtoS}	10 M $\cdot$ s^{-1}, 8 M $\cdot$ s^{-1}	Transfer bandwidth between edge servers and from mobile devices to edge services

Table 2: Slice results for the task workflow by different methods.

Task slicing method	Task slice results
HPD0	$\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}\}$
HPD1	$\{t_1\}\{t_2, t_4, t_5, t_6, t_9, t_{10}, t_{11}\}\{t_3, t_7, t_8\}$
HPD2	$\{t_1, t_2, t_5, t_8, t_{10}, t_{11}\}\{t_4, t_7, t_9\}\{t_3, t_6\}$
HPD3	$\{t_1, t_2, t_5, t_8, t_{11}\}\{t_4, t_7, t_{10}\}\{t_3, t_6, t_9\}$
HIPD0	$\{t_1, t_2, t_3, t_5, t_6, t_7, t_8, t_{10}, t_{11}\}\{t_4\}\{t_9\}$
HIPD1	$\{t_1, t_9\}\{t_2, t_3, t_5, t_6, t_7, t_8, t_{10}, t_{11}\}\{t_4\}$
HIPD2	$\{t_1, t_4, t_7\}\{t_2, t_5, t_8, t_{10}\}\{t_3, t_6, t_9, t_{11}\}$
HIPD3	$\{t_1, t_2, t_5, t_8, t_{11}\}\{t_4, t_7, t_{10}\}\{t_3, t_6, t_9\}$
SSCS	$\{t_1, t_3, t_6, t_7\}\{t_2, t_5, t_8, t_{10}, t_{11}\}\{t_4, t_9\}$

during the execution process of the task, and the idle time of the edge server. In order to better prove the stability and applicability of our algorithm, in the experiments, we assigned two load schemes under different conditions to the subtasks in Figure 5. In the first case, all the subtasks are executed only once. The experimental comparison results are shown in Figures 6(a)–6(c). In the second case, all the subtasks are executed many times, and some subtasks may not be executed. The results of experimental comparison are shown in Figures 6(d)–6(f). Figures 6(a)–6(c) show the overall workflow computation delay corresponding to different slice results in Table 2, the average data transfer load between different edge servers, and the average idle time of the server itself during the execution process. It can be seen from these three figures that, compared with other methods, the SSCS method has the shortest total delay of task workflow, lower average data transmission between subtasks introduced by task distribution deployment, and the highest server utilization rate. Different task slicing schemes obtained by HPD and HIPD methods are effective and shorten the overall delay of task workflow, but they usually introduce a large amount of task transmission. At the same time, they are not superior to our method in terms of server utilization.

For example, the traffic volume introduced by HPD1, HIPD0, and HIPD1 methods is much more than that of our method. For the HPD2 and HIDP2 algorithms, their goal is to slice tasks more evenly across complex workflows. Although in the long run they can make the load distributed among the different server edges more balanced, in most cases, the total delay of the tasks they generate is higher than that of our method. Figures 6(d)–6(f) show that the SSCS method is applicable to different task loads of different sizes. In this case, the slice deployment using the SSCS method can obtain the lowest task computation time delay. It also has good load balance and server utilization. This lays a good foundation for the choreography of subtasks.

5.2. Verification Experiments of Choreography Method. In this group of experiments, we verified the effect of our choreography method from the overall feedback delay of the task workflow, as well as the number and time of completions of the subtasks on the edge servers at each time period.

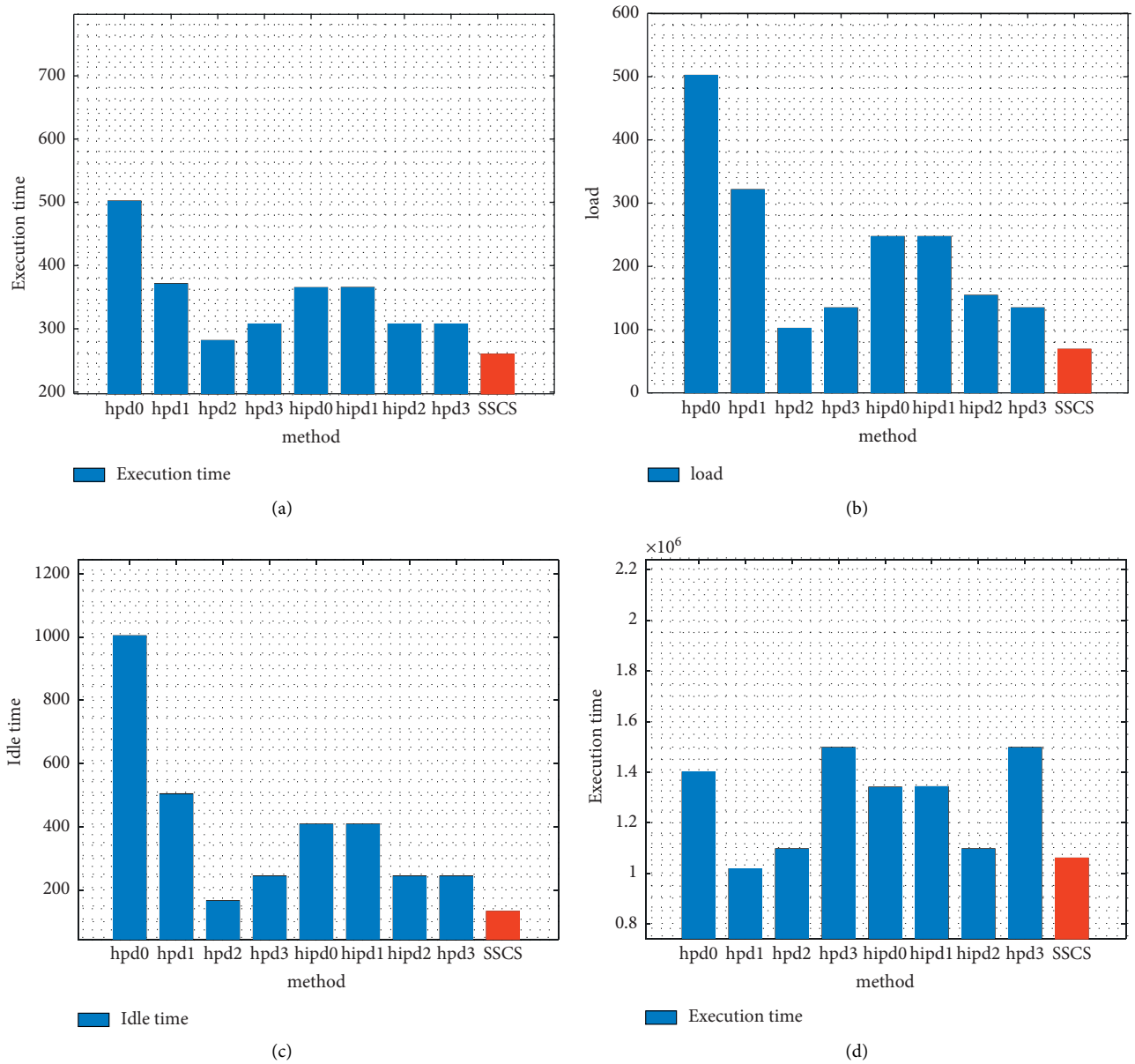


Figure 6: Continued.

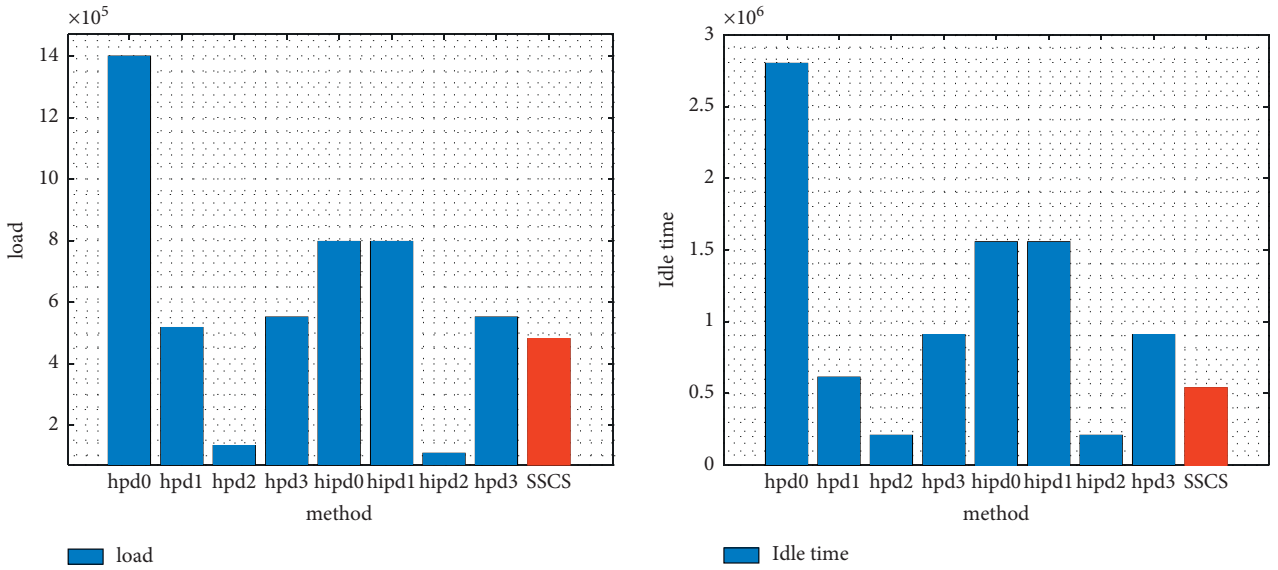


Figure 6: Comparison experiments based on the results of task slices in Table 1 under different task loads. (a) Comparison of execution time of edge server of each method. (b) Load balancing. (c) Edge server free time comparison. (d) Execution time of the task workflow after 10000 executions. (e) Load balance (10000 executions). (f) Idle time on the edge server (10000 executions).

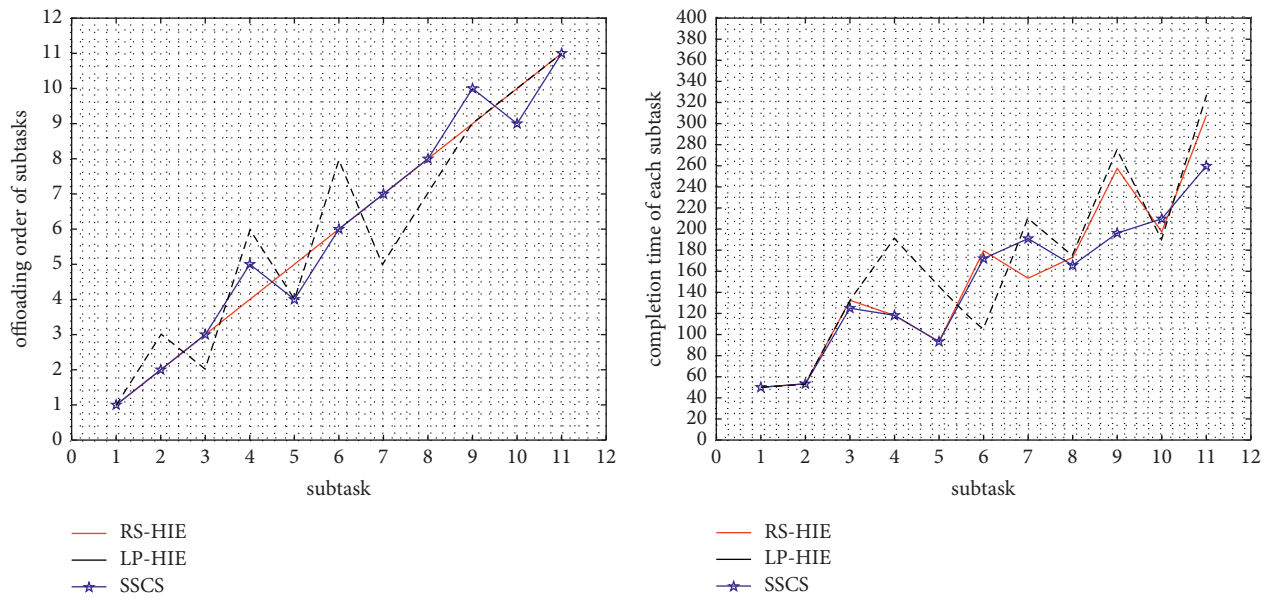
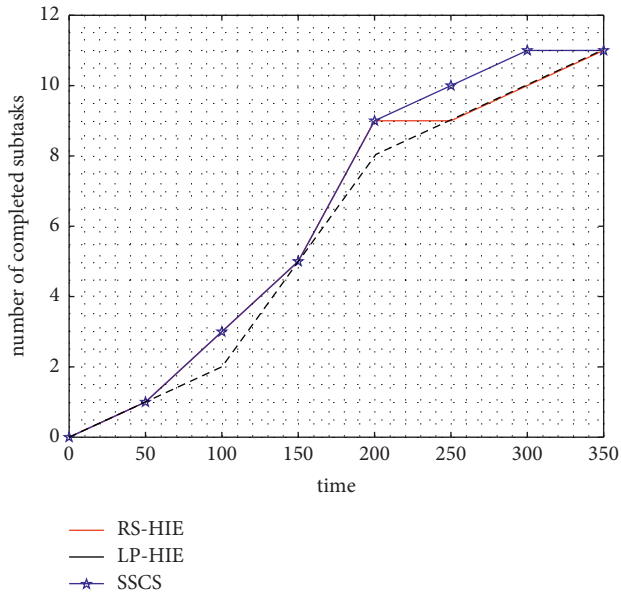


Figure 7: Continued.



(c)

Figure 7: Comparison experiments based on different task choreography methods.

We choose two common task offloading sorting methods. One is according to the hierarchical structure tree of the task workflow; the lower levels of subtasks are first offloaded and the same levels of subtasks are randomly ordered, which is called random sorting based on hierarchy (RS-HIE). The other is similar idea but for the same levels of subtasks which are ordered according to the size of the load, called load prioritization based on hierarchy (LP-HIE). The three offloading sequencing methods are completed based on the task slice results of our method. The experimental results are shown in Figure 7. Figure 7(a) corresponds to the offloading order of subtasks solved by the three methods. It can be seen that the offloading order of subtasks generated by different methods is greatly different. Figures 7(b) and 7(c) show the impact of different offloading sequences on the completion time of subtasks on the edge servers. Figure 7(b) shows the completion time of each subtask in the task workflow on the edge servers, and Figure 7(c) shows the number of subtasks that have been completed on the edge servers in each period. As you can see, the SSCS method can offload the most subtasks per unit time and has the shortest total computation time. It gives full consideration to various key time points affecting the task workflow when solving the choreography scheme, reasonably arranges the offloading time of each subtask, and avoids the total delay caused by waiting for the task to be offloaded as much as possible. Some critical tasks that affect the overall latency of the workflow are offloaded first.

6. Conclusion

To resolve offloading problem of complex tasks in the MEC system, we study the distributed deployment strategy and efficient offloading method of the task workflow. Considering the characteristics of data distribution and logic relationships between subtasks in the process of task offloading and execution, a slicing method of task distribution and deployment is proposed based on similarity between subtasks, and a choreographing method of task offloading sequence is proposed based on the longest overlapping path. Finally, aiming at minimizing the overall delay of task workflow, a heuristic algorithm is designed to solve the approximate optimal solution of slicing and choreographing. Simulation experiments compare our method with other commonly used slicing and choreography methods and prove the effectiveness and advantages of our method in solving task offloading of complex tasks from various angles.

Data Availability

The data involved in this paper include migration algorithm code and simulation data (generated by the simulation algorithm). For copyright protection purposes, data access currently requires contact with institutional authors. If data are needed, the authors should be e-mailed at 55234043@qq.com.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 62162047 and 62162046; Natural Science Foundation of Inner Mongolia under Grants 2019ZD15 and 2019MS06029; Inner Mongolia Science and Technology Plan Project under Grants 2021GG0155 and 2019GG372; the Self-topic of Engineering Research Center of Ecological Big Data, Ministry of

Education; the Open-topic of Inner Mongolia Big Data Laboratory for Discipline Inspection and Supervision (IMDBD2020012 and IMDBD2021014); Inner Mongolia Engineering Laboratory for Cloud Computing and Service Software; Inner Mongolia Key Laboratory of Social Computing and Data Processing; and Inner Mongolia Engineering Lab of Big Data Analysis Technology.

References

[1] Y. Zhao, L. Jia, Y. Zhang, Y. Song, and Y. Tian, "Ordinal multi-task part segmentation with recurrent prior generation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1636–1648, 2019.

[2] M. Fayyaz, B. Cao, W. Almughalles, Y. Li, and L. Ali, "Optimizing task execution for mobile edge computing," in *Proceedings of the 2019 The 8th International Conference on Network, Communication and Computing*, pp. 13065–13076, Luoyang, China, December 2019.

[3] L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu, "Multi-server multi-user multi-task computation offloading for mobile edge computing networks," *Sensors*, vol. 19, no. 6, p. 1446, 2019.

[4] T. Mahn, H. Al-Shatri, and A. Klein, "Distributed algorithm for energy efficient joint cloud and edge computing with splittable tasks," in *Proceedings of the IEEE Conference on Wireless Communications and Networking*, pp. 167–182, Marrakesh, Morocco, April 2019.

[5] K. Li, "Optimal task dispatching on multiple heterogeneous multiserver systems with dynamic speed and power management," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 167–182, 2017.

[6] H. Liu, J. Pu, L. T. Yang et al., "A holistic optimization framework for mobile cloud task scheduling," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 2, pp. 217–230, 2017.

[7] P. Wang, Z. Zheng, B. Di, and L. Song, "Joint task assignment and resource allocation in the heterogeneous multi-layer mobile edge computing networks," in *Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM)*, vol. 69, no. 3, pp. 217–230, IEEE, Waikoloa, HI, USA, December 2019.

[8] K. Li, "Energy-efficient task scheduling on multiple heterogeneous computers: algorithms, analysis, and performance evaluation," *IEEE Transactions on Sustainable Computing*, vol. 1, no. 1, pp. 7–19, 2016.

[9] A. Mukherjee, S. Misra, V. S. P. Chandra, and N. S. Raghuwanshi, "ECoR: energy-aware collaborative routing for task offload in sustainable UAV swarms," *IEEE Transactions On Sustainable Computing*, vol. 5, no. 4, pp. 514–525, 2020.

[10] H. Xing, L. Liu, J. Xu, and A. Nallanathan, "Joint task assignment and wireless resource allocation for cooperative mobile-edge computing," in *Proceedings of the 2018 IEEE International Conference on Communications*, vol. 20, no. 1, pp. 360–374, Kansas City, MO, USA, May 2018.

[11] K. A. Noghani, H. Ghazzai, and A. Kassler, "A generic framework for task offloading in mmWave MEC backhaul networks," in *Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM)*, Abu Dhabi, UAE, December 2018.

[12] Y. Sun, T. Wei, H. Li, Y. Zhang, and W. Wu, "Energy-efficient multimedia task assignment and computing offloading for mobile edge computing networks," *IEEE Access*, vol. 8, pp. 36702–36713, 2020.

[13] M. Tajallifar, S. Ebrahimi, M. R. Javan, N. Mokari, and L. Chiaraviglio, "Energy-efficient task offloading under E2E latency constraints," *IEEE Transactions on Green Communications & Networking*, 2021.

[14] X. Lyu, T. Hui, C. Sengul, and Z. Ping, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 4, pp. 3435–3447, 2016.

[15] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.

[16] T. Yang, R. Chai, and L. Zhang, "Latency optimization-based joint task offloading and scheduling for multi-user MEC system," in *Proceedings of the 2020 29th Wireless and Optical Communications Conference (WOCC)*, Newark, NJ, USA, May 2020.

[17] J. Xue, Y. An, and Y. An, "Joint task offloading and resource allocation for multi-task multi-server NOMA-MEC networks," *IEEE Access*, vol. 9, pp. 16152–16163, 2021.

[18] Y. Zeng, W. Chen, Z. Tang, and J. Wu, "Joint proportional task offloading and resource allocation for MEC in ultra-dense networks with improved whale optimization algorithm," *Journal of Physics: Conference Series*, vol. 1646, no. 1, 2020.

[19] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digital Communications and Networks*, vol. 5, no. 1, pp. 10–17, 2019.

[20] P. A. Apostolopoulos, E. E. Tsiropoulou, and S. Papavassiliou, "Risk-aware data offloading in multi-server multi-access edge computing environment," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1405–1418, 2020.

[21] M. Anedda, C. Desogus, M. Murroni, D. D. Giusto, and G. Muntean, "An energy-efficient solution for multi-hop communications in low power wide area networks," in *Proceedings of the 2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–5, Valencia, Spain, June 2018.

[22] S. Cheng, Z. Chen, J. Li, and H. Gao, "Task assignment algorithms in data shared mobile edge computing systems," in *Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, Dallas, TX, USA, July 2019.

[23] D. Nan, N. S. Hang, X. U. Li, and G. Z. Tan, "Data related task scheduling for vehicular ad hoc networks," *Chinese Journal of Computers*, vol. 40, no. 7, pp. 1614–1625, 2017.

[24] U. Saleem, Y. Liu, S. Jangsher, Y. Li, and T. Jiang, "Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 360–374, 2021.

[25] Z. Wang, Z. Zhao, G. Min, X. Huang, Q. Ni, and R. Wang, "User mobility aware task assignment for mobile edge computing," *Future Generation Computer Systems*, vol. 85, pp. 1–8, 2018.

[26] C. Zhu, J. Tao, G. Pastor et al., "Folo: latency and quality optimized task allocation in vehicular fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4150–4161, 2019.

[27] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, pp. 26652–26664, 2019.

[28] X. Wang, Y. Cui, Z. Liu, J. Guo, and M. Yang, "Optimal resource allocation for multi-user MEC with arbitrary task arrival times and deadlines," in *Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, May 2019.

[29] M. Emara, H. ElSawy, M. C. Filippou, and G. Bauch, "Spatio-temporal dependable task execution services in MEC-enabled wireless systems," *IEEE Wireless Communications Letters*, vol. 10, no. 2, pp. 211–215, 2021.

[30] X. Long, J. Wu, Y. Wu, and L. Chen, "Task merging and scheduling for parallel deep learning applications in mobile edge computing," in *Proceedings of the 2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Gold Coast, Australia, December 2019.

[31] A. Mamat, Y. Lu, J. Deogun, and S. Goddard, "Efficient real-time divisible load scheduling," *Journal of Parallel and Distributed Computing*, vol. 72, no. 12, pp. 1603–1616, 2012.

[32] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," 2016, https://arxiv.org/abs/1604.07525.

[33] G. Wu, Y. Li, J. Ren, and C. Lin, "Partitioned fixed-priority real-time scheduling based on dependent task-split on multicore platform," in *Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, Melbourne, Australia, May 2013.

[34] Z. Liu, X. Yang, and J. Shen, "Optimization of multitask parallel mobile edge computing strategy based on deep learning architecture," *Design Automation for Embedded Systems*, vol. 24, no. 3, pp. 129–143, 2020.

[35] Z. Xie, X. Song, and S. Xu, "Peer-to-peer enhanced task scheduling for D2D enabled MEC network," *IEEE Access*, vol. 8, pp. 138236–138250, 2020.

[36] F. Safi Esfahani, M. A. A. Murad, M. N. Sulaiman, and N. I. Udzir, "Using process mining to business process distribution," in *Proceedings of the 2009 ACM Symposium on Applied Computing*, pp. 2140–2145, Honolulu, HI, USA, March 2009.

[37] F. Safi Esfahani, M. A. A. Murad, M. N. B. Sulaiman, and N. I. Udzir, "Adaptable decentralized service oriented architecture," *Journal of Systems and Software*, vol. 84, pp. 159l–1617, 2011.