Journal of Computers Vol. 34 No. 5, October 2023, pp. 149-166
doi: 10.53106/199115992023103405011

A Multi-edge Collaborative Computational Offloading Scheme Based on Game Theory

Qiao-Feng Song[1], Jun Wang[1,2*], Ji-Xu Gao[1], Jia-Hao Liu[1]

[1] School of Communication and Information Engineering, Nanjing University of Posts and Telecommunications
Nanjing 210003, China
694398201@qq.com

[2] Jiangsu Key Laboratory of Wireless Communication, Nanjing University of Posts and Telecommunications
Nanjing 210003, China
wang_jun@njupt.edu.cn

Received 15 October 2022; Revised 21 March 2023; Accepted 1 May 2023

Abstract. A common approach in existing collaborative edge computing offloading schemes is to partition tasks into independent sub-tasks and offload them to participating servers. However, in practice, these sub-tasks often have dependencies, resulting in waiting time. To address this problem, we propose a collaborative computation offloading scheme based on Stackelberg game theory and graph theory (CCOSGG). First, we introduce a task clustering method based on graph theory, which uses task reconstruction and graph partition algorithm to cluster strongly related sub-tasks into appropriately sized clusters. Second, we use Stackelberg game theory to introduce an incentive mechanism that encourages remote edges to participate in the collaborative offloading. Finally, simulation results demonstrate that the proposed scheme can minimize latency and energy consumption at different network scales.

Keywords: edge computing (EC), computation offloading, multi-edge collaborative, stackelberg game, graph theory

1 Introduction

The rapid advancement of smart devices and 5G communication technology has given rise to various demanding applications, such as autonomous driving, virtual and augmented reality, live video streaming, and interactive gaming. These applications require high computing resources to deliver accurate and timely services [1]. Devices do not have enough computing resources to meet user requirements, resulting in an inability to satisfy low-latency demands [2]. To address this issue, mobile edge computing (MEC) has emerged as a new solution. By leveraging edge servers, which are closer to devices than the cloud center and have ample computing resources, MEC can meet low-latency requirements effectively.

The task offloading problem has become a key area of research in MEC, as it enables effective utilization of computing resources. Its core objective is to optimize task offloading decisions based on task and computing resource characteristics [3]. Current research in computing offloading mainly focuses on optimized collaborative strategies to minimize system cost. Collaborative computing offloading aims to reduce the computing burden on terminal devices by offloading tasks to edge servers or the cloud center [4]. This decreases energy consumption and latency and enhances task execution efficiency and quality. To achieve this, tasks are usually divided into multiple sub-tasks and processed in parallel by multiple servers.

Most existing research assumes that sub-tasks are independent, ignoring the fact that there may be dependencies between them. In reality, some sub-tasks rely on the output of others as their input [5], leading to waiting time that can increase total task latency and negatively impact user experience. Therefore, the key challenge we aim to address is how to reduce the waiting time between sub-tasks with dependencies.

The main research contributions of this paper can be summarized as follows:

Firstly, an improved task clustering method based on graph theory is proposed to effectively reduce the waiting delay between sub-tasks that have dependencies.

Secondly, system latency and energy consumption can be reduced by combining horizontal collaboration be-

* Corresponding Author

tween edge servers and vertical collaboration between edge servers and the cloud center.

Finally, a reward mechanism is introduced through game theory to encourage active participation of remote edge servers in collaborative offloading. The Nash equilibrium between non-cooperative games is then solved by reverse induction.

This paper is organized as follows. Section 2 introduces the related work of related algorithms. The system model and the problem formulation are described in Section 3. In Section 4, the CCOSGG are described in detail. Then, we discuss the simulation results in Section 5. Finally, conclusions are drawn in Section 6.

2 Related Works

In recent years, research on collaborative computing offloading has mainly focused on two aspects: vertical collaboration and horizontal collaboration. Vertical collaboration refers to collaboration between different layers in a three-layered computing architecture, such as cloud-edge collaboration or cloud-edge-device collaboration. Li et al. [6] designed a computation offloading and resource allocation policy based on task priority and task deadline to minimize computation offloading latency, but this scheme based on cloud-edge collaboration ignored the computing resources of devices in the architecture. To address this, Refs [7-10] proposed a cloud-edge-device collaboration architecture. Wu et al. [7] took into account the inter-cell interference induced by using the same channel to communicate, and established an optimization problem with the objective of minimizing the weighted sum of system latency and energy consumption. As each user's offloading decision depends not only on its own strategy but also on the strategies of other users, they solved the optimization problem using game theory methods. Kai et al. [8] proposed a pipeline-based offloading scheme, where both mobile devices and edge nodes can offload computation-intensive tasks to a particular edge server or the cloud center, and they adopted the classic successive convex approximation (SCA) approach to solve the optimization problem in this paper. Hossain et al. [9] proposed a dynamic offloading decision scheme, which adaptively chooses cloud-edge collaboration or edge-device collaboration according based on different scenarios. Simulation results confirm that compared to other schemes, this approach can effectively reduce the average number of task failures and provide lower execution latency.

The above-mentioned works fully utilize the available resources in the vertical direction of the three-layer architecture of edge computing, but they overlook the available resources in each layer of the architecture, such as idle devices in the device layer and adjacent edge servers with light workloads in the edge server layer. To address this issue, it is necessary to introduce horizontal collaboration to complement the vertical collaboration paradigm. Song et al. [11] proposed a collaborative offloading strategy based on vehicular networks, where computing tasks can be processed locally, offloaded to roadside units or the cloud center, and also can be offloaded to cooperative vehicles for processing. Simulation results show that this approach can provide sufficient computing resources for vehicles. Qian et al. [12] designed a novel Device-to-Device (D2D) collaborative offloading method based on game theory, which introduces a reward mechanism to encourage idle devices to participate in offloading.

Compared to devices, edge servers have more powerful computing capabilities and faster communication speeds. Therefore, some studies have introduced collaboration between edge servers to further improve offloading efficiency. Gu et al. [13] proposed an AUC-AC algorithm for resource allocation of channel and computing, which is based on the dominant actor-critic network and the auction mechanism. This scheme includes edge-edge collaboration and edge-cloud collaboration. To make multiple edge servers or the cloud center serve as many UEs as possible, Tian et al. [14] designed a response ratio offloading strategy (RROS) centered on user preference and real-time nature. Lin et al. [15] considered the computing networks where edge nodes work collaboratively to provide efficient computation services to terminals, and the drift-plus-penalty Lyapunov optimization approach is used to solve the optimization problem. Numerical results have shown that this collaborative strategy has better performance in reducing system latency.

Offloading strategies in the above works are binary, meaning that the entire task is offloaded to the target server. While partial offloading can divide the task into multiple sub-tasks and process them in parallel with collaborative servers, resulting in better reduction of task response time. Partial offloading has also been investigated in the existing works such as [16-20]. However, these works took no account of the topology of the task. Practically, the output of some sub-tasks is the input of others, so we cannot ignore the dependencies between sub-tasks. The dependencies among sub-tasks can result in waiting delays, which increase the total system latency. Therefore, it is necessary to propose a reasonable sub-tasks clustering algorithm to reduce the waiting delay, which is also a key point of this study.

3 System Model and Problem Formulation

As shown in Fig. 1, the system architecture of our scheme consists of user equipment (UE), the local mobile edge server (LM), remote mobile edge servers (RM) and the cloud center (CC). In this architecture, the server closest to the UE is considered as the local edge server. Due to the limited computing resources of IoT nodes, we assume that the UE's computing capacity can be directly ignored. All computing tasks from IoT nodes will be offloaded to the local edge server, and then the local edge server will make offloading decisions to determine whether tasks need to be further offloaded to remote edges or the central cloud for execution.

Assuming that all computational tasks generated by UE can be partitioned into K sub-tasks based on their dependencies, these sub-tasks can be represented as $K = \{1, 2, 3, \ldots, k\}$. Each task can be denoted as $k = (b_k, W_k)$, where b_k is the data size of sub-task k and W_k represents the amount of computing resources (the number of CPU cycles) required to execute this sub-task. Remote edge nodes near the local edge are denoted as $N = \{1, 2, 3, \ldots, n\}$.

Due to the existence of dependency relationships between sub-tasks, the execution of each sub-task is contingent upon the completion of its predecessors. If a predecessor sub-task is not completed, subsequent sub-tasks cannot proceed.

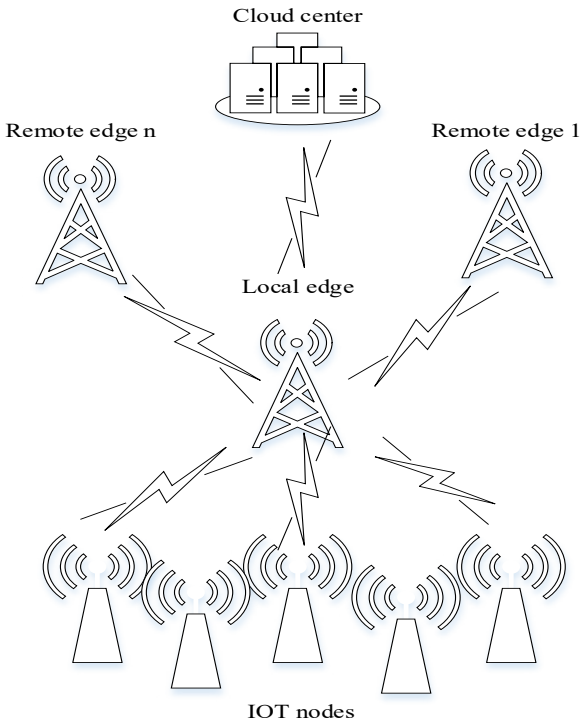

Fig. 1. The model of collaborative offloading for completely offloading scenarios

3.1 Communication Model

Data transmission between UE and LM typically occurs through wireless media such as 4G, 5G, Wi-Fi, and Bluetooth, the transmission rate between them can be expressed as:

$$V_{U,L} = B_{U,L} \log_2 (1 + \frac{H_{UE} G_{U,L}}{\sigma^2}). \quad (1)$$

where $B_{U,L}$ is the channel bandwidth between UE and LM, H_{UE} is the transmit power of UE, $G_{U,L}$ denotes the channel gain between UE and LM, σ^2 is the Gaussian noise power. This paper assumes that the channels between UEs are mutually orthogonal, so the signal interference between UEs can be ignored.

The local edge and remote edge servers are small cell cloud (SCC) with the same system architecture, but differ only in their distance from the UE. As they are usually interconnected by wired optical cables, the link transmission rate between the local edge server and each remote edge server can be set as the same fixed value $V_{L,Rn} = V_c$, which is determined after installation of the equipment.

For the link transmission rate from the LM to the CC, it can be described as

$$V_{L,C} = B_{L,C} \log_2 \left(1 + \frac{H_{LC} G_{L,C}}{\sigma^2} \right). \tag{2}$$

Where $B_{L,C}$ is the channel bandwidth between LM and CC, H_{LC} is the transmit power of the LM, $G_{L,C}$denotes the gain between LM and CC.

3.2 Computation Model

In the proposed collaborative offloading model in this paper, the tasks generated by UE are first divided into K sub-tasks, which are then all transmitted to the local edge server. The resulting transmission delay can be expressed as:

$$T_{U,L}^k = \frac{b_k}{V_{U,L}} \quad \forall_k \in K. \tag{3}$$

where b_k is the data size of the task k, and $V_{U,L}$ denotes the transmission rate between UE and the LM.

The corresponding transmission energy consumption can be represented as:

$$E_{U,L}^k = b_k \gamma_{U,L}. \tag{4}$$

where $\gamma_{U,L}$ is the energy consumption for transmitting per unit size of task from UE to the LM.

For these sub-tasks, the local edge server can choose to execute them locally or offload them to remote edge servers or the cloud center. We will discuss the corresponding time delay and energy consumption for executing sub-tasks on the local edge server, remote edge servers, and the cloud center. Let binary variable $x_k = \{0,1\}$, $y_k^n = \{0,1\}$, $z_k = \{0,1\} \forall_k \in K$ as offloading decision of sub-task k, where $x_k = 1$, $y_k^n = 1$ and $z_k = 1$ indicates that the task is executed on the LM, RM n and the CC respectively.

1) sub-task k is executed locally on the LM

The execution time using the LM CPU is given by:

$$T_L^k = \frac{W_k}{f}. \tag{5}$$

where f_L is computing capacity of the LM, and W_k is the amount of computing required for executing sub-task k.

Energy consumption of computing sub-task k for the LM is:

$$E_L^k = b_k e_L \ . \tag{6}$$

where e_L represents the energy consumption required by the LM to process per unit data of task.

2) sub-task k is offloaded to remote edge server n

Firstly, the transmission latency for transmitting the data of sub-task k from the LM to RM n is represented as:

$$T_{L,R_n}^k = \frac{b_k}{V_{L,R_n}} = \frac{b_k}{V_c} \ . \tag{7}$$

The corresponding transmission energy consumption is:

$$E_{L,R_n}^k = b_k \gamma_{L,R} \ . \tag{8}$$

where $\gamma_{L,R}$ is the energy consumption for transmitting per unit size of task from the LM to RM n.

RM and LM have the same structure, so RM's computing capacity is also f_L. Then, the execution time of computing sub-task k for RM n is:

$$T_{R_n}^k = T_L^k = \frac{W_k}{f_L} \ . \tag{9}$$

And energy consumption of computing sub-task k also can be represented as:

$$E_{R_n}^k = E_L^k = b_k e_L . \tag{10}$$

3) sub-task k is offloaded to the CC

Similarly, offloading sub-task to the cloud center also includes two processes: data transmission and computation.

The transmission latency for transmitting the data of sub-task k from the LM to the CC is:

$$T_{LC}^k = \frac{b_k}{V_{L,C}}. \tag{11}$$

where $V_{L,C}$ denotes the transmission rate between the LM and the CC.

Energy consumption of transmitting data is given as:

$$E_{L,C}^k = b_k \gamma_{L,C}. \tag{12}$$

where $\gamma_{L,C}$ is the energy consumption for transmitting per unit size of task from the LM to the CC.

The computing capacity of the CC is f_c, so the execution time of computing sub-task k for the CC is:

$$T_C^k = \frac{W_k}{f_c}. \tag{13}$$

The corresponding energy consumption of computing sub-task can be expressed as:

$$E_C^k = b_k e_C. \tag{14}$$

where e_C represents the energy consumption required by the CC to process per unit data of task.

As there are interdependencies between sub-tasks, the execution of a given sub-task k is contingent on the completion of all its predecessor tasks. Accordingly, the waiting time of sub-task k is defined as the maximum of the execution latency of all its predecessors. Since the resulting data size is very small, we can ignore the feedback time of the computation result. Therefore, the waiting time for sub-task k can be expressed as:

$$T_k^{wait} = \max_{m \in pred(k)} \max\{T_L^m, T_{R_n}^m, T_c^m\}. \tag{15}$$

where *pred*(*k*) denotes the set of associated tasks that need to be completed before sub-task *k* can be executed.

Therefore, the total latency of sub-task *k* is the sum of transmission delay, execution delay and waiting time. It can be given as:

$$T_{sum}^k = T_{U,L}^k + x_k T_L^k + (1-x_k)[\sum_{n=1}^{N} y_k^n (T_{L,R_n}^k + T_{R_n}^k) + z_k (T_{L,C}^k + T_C^k)] + T_k^{wait}. \tag{16}$$

The total energy consumption of sub-task *k* is:

$$E_{sum}^k = E_{U,L}^k + x_k E_L^k + (1-x_k)[\sum_{n=1}^{N} y_k^n (E_{L,R_n}^k + E_{R_n}^k) + z_k (E_{L,C}^k + E_C^k)]. \tag{17}$$

3.3 Problem Formulation

The optimization goal in this paper is a weighted sum of delay and energy consumption. so the objective function of the whole system is:

$$\sum_{k=1}^{K} (\alpha T_{sum}^k + (1-\alpha) E_{sum}^k). \tag{18}$$

where α is balance factor to control the proportion of energy consumption and latency in total costs. The corresponding constraint are:

$$\sum_{k=1}^{K} E_{U,L}^k \le E_{U,max} \tag{19}$$

$$\sum_{k=1}^{K} \sum_{n=1}^{N} E_L^k + y_k^n E_{L,R_n}^k + z_k E_{L,C}^k \le E_{L,max} \tag{20}$$

$$\sum_{k=1}^{K} y_k^n E_{R_n}^k \le E_{R_n,max} \tag{21}$$

$$V_{U,L} \le V_{L,R_n} + V_{L,C} \tag{22}$$

$$x_k + \sum_{n=1}^{N} y_k^n + z_k = 1\,. \tag{23}$$

where the constraint (19) indicates that the energy consumption of transmitting all sub-tasks from UE should be less than its own total energy, ensuring that all sub-tasks can be transmitted successfully. The constraint (20) indicates that the total energy consumption of executing sub-tasks and transmitting sub-tasks for the LM should be less than its own total energy. The constraint (21) indicates that the energy consumption of executing sub-tasks for RM *n* should be less than its own total energy. The CC is generally considered to have infinite energy, so there is no energy constraint for it. The constraint (22) indicates that the sum of transmission rate from the LM to RM *n* and transmission rate from the LM to the CC is greater than transmission rate from UE to the LM, which ensures that there is no task backlog after sub-tasks have been offloaded to the LM.

4 An Offloading Scheme Based on Graph Theory and Game Theory

To reduce the waiting time of sub-tasks, an improved task clustering algorithm based on graph theory is proposed. Initially, the algorithm utilizes task clustering to establish distinct sub-task clusters based on their dependence relationships. However, following classical topological sorting, these clusters may exhibit a large number of sub-tasks, leading to the centralization of processing and negating the benefits of task partitioning. To address this, the algorithm employs a graph partitioning algorithm to subdivide separable sub-task clusters into smaller clusters, thereby ensuring the distribution of sub-tasks across different servers and improving the efficiency of distributed computing.

Secondly, to encourage remote edge servers to participate in collaborative offloading, a differentiated pricing collaborative computing offloading algorithm based on Stackelberg game theory is proposed in this section. This algorithm introduces a reward mechanism to encourage remote edge servers to participate in collaboration. And then, backward induction is used to solve the game problem and optimal offloading decision will be obtained.

4.1 An Improved Task Clustering Algorithm

Firstly, sub-tasks can form a Directed Acyclic Graph (DAG). Each vertex in the graph represents a task node. In a DAG, if sub-task T1 depends on sub-task T2, that means sub-task T1 should be executed after sub-task T2. In the graph, T1 will have an edge connecting to T2.

To reduce the waiting time of sub-tasks, topological sorting (TS) is performed on the directed acyclic graph (DAG) to ensure that all task nodes are ordered according to their dependencies. TS guarantees that parent task nodes do not depend on any child task nodes. However, it has the following drawbacks: as shown in Fig. 2, the two subgraphs above are sub-task sets obtained by classical TS. In the left subgraph, the execution of sub-task T1 depends on the output of sub-task T2, and in the right subgraph, the execution of sub-task T1 depends on the output of sub-task T5, T6, and T7. After offloading the two clusters to separate servers, computing T1 in the right subgraph requires receiving the result of T5, T6, and T7. At the same time, computing T1 in the left subgraph also requires receiving the result of T2. This increases the waiting time of sub-task T1, which is not desirable.

In order to deal with the complex relationship between sub-tasks, an improved topological sorting (TS) algorithm has been proposed to group associated sub-tasks into subgraphs. As shown in Fig. 2, two sub-task sets that were previously obtained through classical TS are re-aggregated into a cluster using the task reconstruction algorithm. In this new sub-task set, the sub-tasks that sub-task T1 depends on are more clearly represented. If all sub-tasks within the same cluster are offloaded to the same edge servers, the transmission delay and energy consumption can be reduced, leading to a decrease in waiting time.

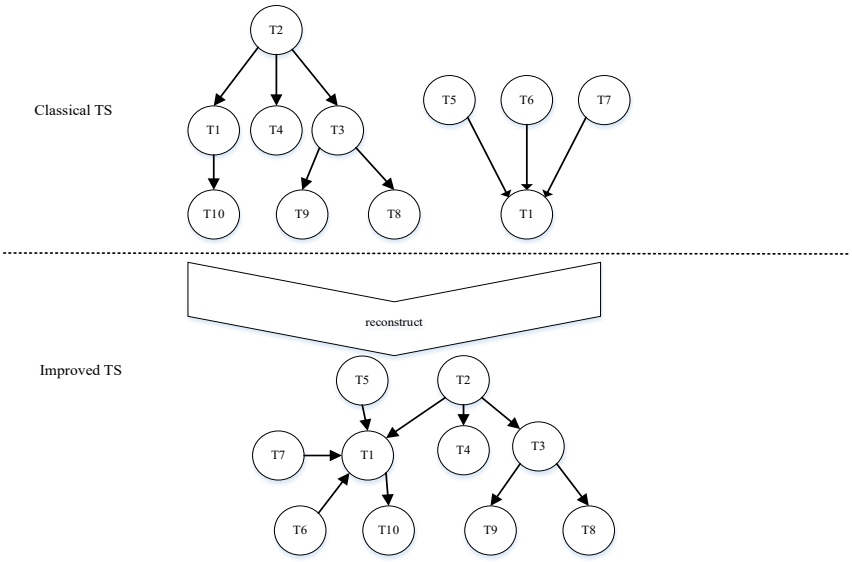

Fig. 2. The model of collaborative offloading for completely offloading scenarios

Each node in the graph is associated with the data size and computation cost. The property of the root node of each subgraph is defined as the sum of the data size and computation cost of all nodes in that subgraph. By traversing the root node of each subgraph, we obtain a list of the computation resources required for all subgraphs. The task reconstruction algorithm is outlined in Algorithm 1.

```
  Algorithm 1.= Task reconstruction algorithm (oldNodeHeadList, newNode-
HeadList)
    visited = 0
    inDegreeMap=NULL, zeroDegreeList=NULL,tempDegreeList = NULL, result-
List = NULL; nodeResultList = NULL
    for all nodes in the set V and if nodes.visited = 0 do
      inDegreeMap.put(v, v.inDegree)
      if v.inDegree == 0 then
        zeroDegreeList.add(v)
        tempDegreeList.put(zeroDegreeList.remove())
        nodeResultList.put(tempDegreeList.remove())
    for tempDegreeList.remove().next do
      new Node from tempDegreeList.remove().next and newNnode. inDegree
-= 1
      inDegreeMap.put(newNode, newNode.inDegree)
      if newNode.indegree = 0,
         tempDegreeList.put(this newNode)
         resultList.addAll(nodeResultList)
end
```

If the computation cost required for the root node of a sub-task cluster after reconstruction is very large, as shown in Fig. 3, it cannot be executed at the edge nodes and must be transferred to the cloud center. This issue can result in too many sub-task clusters being transferred to the cloud, which can undermine the edge-cloud collaboration and waste the computation capability of the edge nodes. To address this problem, the graph partition algorithm is used. The algorithm partitions a graph, represented as $G = (V, E)$, into two non-empty sub-sets E_1 and E_2, where $G[E_1]$ and $G[E_2]$ have only one common vertex V. The vertex V is then a cut vertex of G.

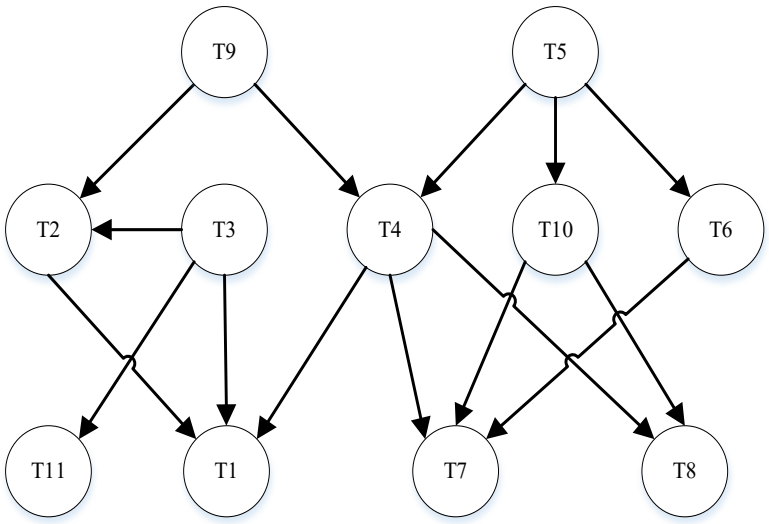

Fig. 3. Task cluster formed by task reconstruction algorithm

Based on Algorithm 2, the left and right sub-graphs shown in Fig. 3 are strongly connected and share a common vertex T4. To decrease the computational cost, the task cluster is divided into two smaller clusters, as illustrated in Fig. 4. These two clusters can now be executed on edge nodes, achieving system load balancing

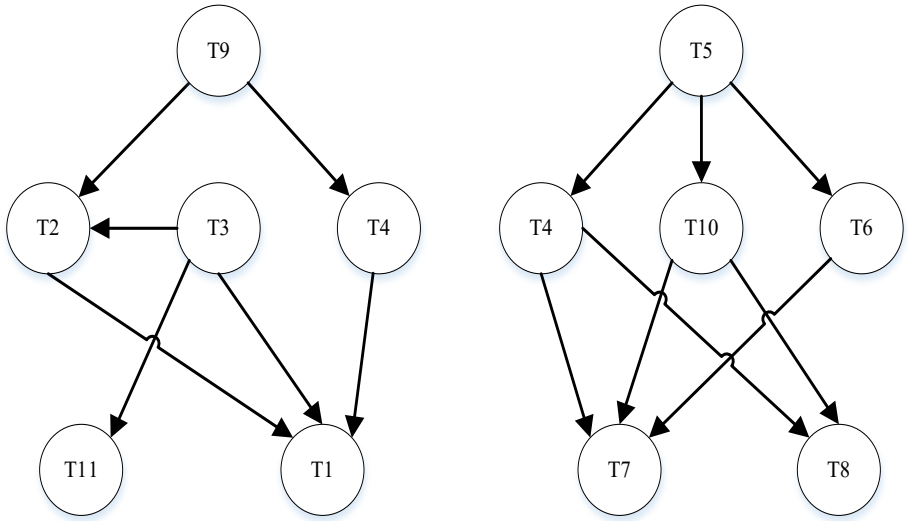

Fig. 4. Task partition

```
  Algorithm 2. Task partition algorithm (oldNodeHeadList after Algorithm 1,
newNodeHeadList after partition)
  new a clusterList
  for nodeResultList in resultList do
      nodeValueSum=0;nodeComSum=0;nodeStack=NULL; nodeSet=NULL
      tempNode = nodeResultList.pop()
      nodeStack.add(tempNode) and nodeSet.add(tempNode)
      compute nodeValueSum and nodeComSum
      currentNode = nodeStack.pop()
      for currentNode.next do
          if currentNode.next is not in nodeSet does then
          put current node and it.next in nodeStack and put it.next in
nodeSet
          compute nodeValueSum and nodeComSum
          if nodeComSum ≥ the servers' computation capacity then
             clusterList.put( NodeHead, nodeValueSum, nodeComSum)
      for set in clusterList do
          start depth-first traversal by referring to step 3 and put each
node's value in Set
          if node in clusterList and Set then
             partition the graph into two subgraphs again
          if nodeComSum ≥ the servers' computation capacity then
             repeat 13 and re-partition by their common vertex
          end if
end
```

4.2 Solve Offloading Decision using Stackelberg Game Theory

In this scenario where full offloading is used, all tasks from the UE should be offloaded to the (LM). The transmission delay and energy consumption during this process are only related to the data size of the tasks, and are not affected by the offloading decisions. Therefore, as illustrated in Fig. 5, the offloading system can be simplified to a two-tier system when making offloading decisions.

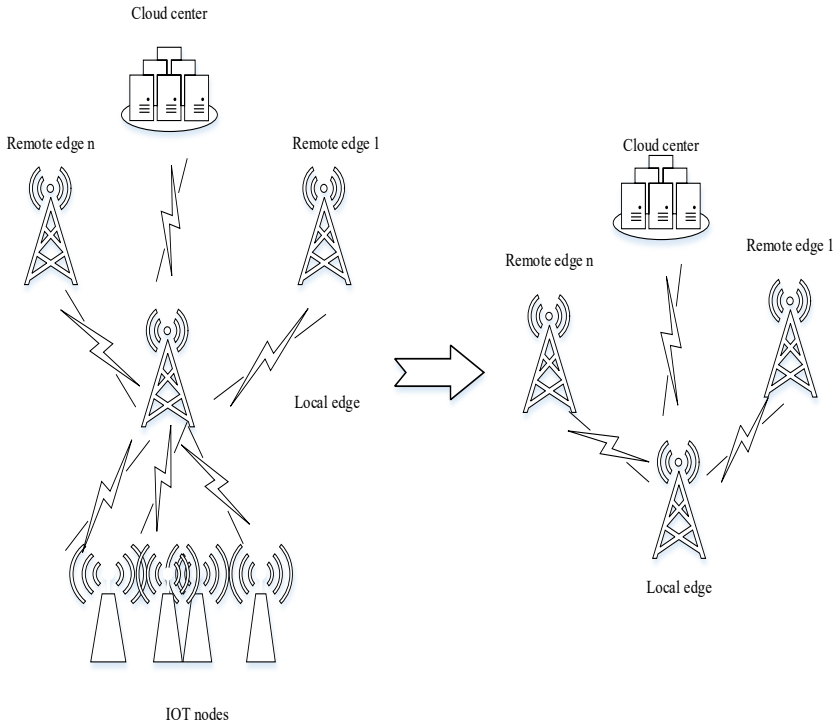

Fig. 5. Simplified two-layer cooperative offloading model

(1) Utility Function

Servers, acting as the leaders, start by setting the price of computing resources for each user. As mentioned in section 3.1, the data size and computational costs of all tasks in a cluster are stored in its root node. This allows the root node k (T9 or T5 in Fig. 4) to represent all nodes in its cluster. The price of per execution unit for user k is set by server j as $p_{j,k}$, and the price for each user is then set as a vector $\{p_{1,1}, p_{1,2}, \ldots, p_{1,k}, \ldots, p_{j,k}\}$. The leader's utility function is determined by the amounts of computational tasks offloaded to it. In this scheme, $x_k = 1$ means that the task is executed at the user, $y_k^n = 1$ means that the task is offloaded to RM n, and $Z_k = 1$ means that the task is offloaded to the CC. The computing resource required for task k is W_k, and the computation cost required at this server is $\sum_{k=1}^{K}(1-x_k)W_k$. Therefore, the utility function of the servers can be written as:

$$\max U = \sum_{j=1}^{N+1}\sum_{k=1}^{K}(1-x_k)W_k p_{j,k}. \tag{24}$$

For the user's side, the system overhead only pertains to the delay, energy consumption, and incentive price. The total delay of each cluster k can be calculated as follows:

$$T = x_k T_L^k + (1-x_k)\left\{\sum_{n=1}^{N} y_k^n (T_{L,R_n}^k + T_{R_n}^k) + z_k (T_{L,C}^k + T_C^k)\right\}. \quad (25)$$

Compared to T_{sum} in Eq. (12), the value of T does not incorporate the transmission latency $T_{U,L}^k$, which is independent of the offloading decision. Furthermore, T does not take into account the waiting time, as clusters become independent of one another after graph partitioning.

The energy consumption of processing is given by:

$$E = x_k E_L^k + (1-x_k)\sum_{n=1}^{N} \{y_k^n (E_{LR}^k + E_R^k) + (1-y_k^n)(E_{LC}^k + z_k E_C^k)\}. \quad (26)$$

Similarly, E in Eq. (28) does not include energy consumption of transmitting tasks from UE to the LM.

To incentivize RMs to participate in the offloading process, an incentive mechanism is introduced where RMs will receive additional rewards that are Q times their original gain. As a result, the utility function of the servers is modified as:

$$\max U = \sum_{j=1}^{N+1}\sum_{k=1}^{K} (1+Qy_k^j)(1-x_k) W_k p_{j,k}. \quad (27)$$

For the user, the cost of offloading tasks to RMs needs to increase by a factor of Q compared to the cost of executing tasks locally. As a result, the utility function of the user is modified as:

$$\max V = -\sum_{j=1}^{N+1}\sum_{k=1}^{K} (\alpha T + (1-\alpha)E + (1-x_k) W_k Q y_k^j p_{j,k}). \quad (28)$$

The game model proposed in this paper is a fully informed dynamic Stackelberg game, consisting of two stages: (i) Leaders (RMs and the CC) set the price p by calculating the requirements information for each user. (ii) Followers (User k) adjust their offloading strategy $(x_k^*, y_k^{n^*}, z_k^*)$ based on servers' price p and their own computing resources' requirements. In this game, each party is selfish and pursues their maximum utility.

2) Backward induction method for solving Stackelberg game

We use backward induction to analyze the Stackelberg game. The leaders (servers) set their price responses (i.e., p^*) by considering the equilibrium reactions of the followers (users) (i.e., $d^*(p^*)$). We start by analyzing the best response of the followers to the assumed optimal price set by the leaders. Then, the leader determines the optimal price based on the predicted responses of the follower. Finally, we obtain the optimal offloading decisions by bringing this optimal price to the responses of the follower. This process is shown in Algorithm 3.

Algorithm 3. Backward induction

assume that server sets the price of computing resource as: $p = \{p_{1,1}, p_{1,2}, \ldots p_{1,k}, \ldots p_{j,k}\}$

solve the maximum of user's utility function, obtain the optimal offloading decision: $d^*(p) = \{(x_1^*, y_1^{n^*}, z_1^*), (x_2^*, y_2^{n^*}, z_2^*), \ldots (x_k^*, y_k^{n^*}, z_k^*)\}$

for $(x_k^*, y_k^{n^*}, z_k^*) \in d^*(p)$ do

    if all of them have unique deterministic value then

        return $d^*(p)$

    end if

```
end for
bring d*(p) into server's utility function to solve servers' price strat-
egy p* at this point
for p*_{j,k} ∈ p* do
    if p_{jk} in the range of resource price then
       return p*
    else
       break
    end if
end for
solve the suboptimal solution of the server's utility function and obtain
the price strategy
repeat the above steps
return the price of computing resource p* and the final optimal offloading
decision d*
end
```

The final solution of the Stackelberg game is the Nash equilibrium [21], where all participants reach their optimal utility values. Once reached, changing one's strategy unilaterally does not result in an increase in utility value.

The user's utility function V_k given by Eq. (30), is a cost function that is minimized by player k. It is defined as $V_k = \alpha T + (1-\alpha)E + (1-x_k)W_k Q y^j_k p_{j,k}$. A Nash equilibrium of the multi-user computation offloading game is a strategy profile $d^*=\{(x^*_1, y^{n*}_1, z^*_1), (x^*_2, y^{n*}_2, z^*_2), ...(x^*_k, y^{n*}_k, z^*_k)\}$, where no user can increase its utility by changing its strategy unilaterally at the equilibrium d^*.

$$V_k((x^*_k, y^{n*}_k, z^*_k),(x^*_{-k}, y^{n*}_{-k}, z^*_{-k})) \le V_k((x_k, y^n_k, z_k),(x^*_{-k}, y^{n*}_{-k}, z^*_{-k})). \quad (29)$$

In the multi-user computation offloading game, if user k chooses the cloud computing approach at Nash equilibrium, it must be beneficial for the user [22].

If cloud computing is not beneficial, the user can improve its benefit by switching to local computing or offloading to RMs, which contradicts the Nash equilibrium definition. Moreover, the Nash equilibrium ensures self-stability, where users can achieve a mutually satisfactory solution [23].

4.3 Complexity of the Algorithm

In the two-stage game, there are $N+1$ servers, and each server computes the best response to maximize its utility based on Eq. (29). Algorithm 3 has the time complexity of $O(KN)$, but after the subtasks are reconstructed and partitioned into $m(m \le K)$, subtasks in Algorithms 1 and 2, the time complexity is reduced to $O(mN)$.

5 Experimental Results and Analysis

To validate the performance of CCOSGG, we constructed a simulation environment for collaborative computing offloading by integrating network conditions and offloading decisions, using the open-source EdgeCloudSim [24] project as a basis.

5.1 Simulation Parameters

To simulate the task generation process realistically, we set the user's simulation parameters as random values that satisfy a uniform distribution. The parameter settings are listed in Table 1:

Table 1. Simulation parameter settings

Simulation parameters	Range of value	Simulation parameters	Range of value
W_k / $\left(\text{cycle}\cdot\text{bit}^{-1}\right)$	200-700	b_k / KB	100-300
H_{UE} / W	0.1-0.5	H_{LM} / W	1-2
$B_{U,L}$ / MHz	5-10	B_{LC} / MHz	10-15
f_L / $(\text{cycle}\cdot\text{s}^{-1})$	5×10^6	f_C / $(\text{cycle}\cdot\text{s}^{-1})$	10×10^6
$\gamma_{U,L}$ / mW	90	e_L / $\left(\text{J}\cdot\text{MB}^{-1}\right)$	1.5
$\gamma_{L,R}$ / mW	200	e_c / $\left(\text{J}\cdot\text{MB}^{-1}\right)$	1
$\gamma_{L,C}$ / mW	400	V_C / $\left(\text{MB}\cdot\text{s}^{-1}\right)$	5

In the simulation, the number of tasks and servers' computing capacity randomly vary to test the system's adaptability to various scenarios. The simulation results demonstrate that the proposed system can effectively handle different workload conditions and provide better performance compared to traditional methods.

5.2 Simulation Results

As shown in Fig. 6, as the reward factor increases, more and more remote edge servers join the system, resulting in the continuous expansion of available computing resources and the gradual reduction of resource's prices. However, when the reward factor reaches a certain value, the resource's price no longer decreases. This is because a large number of remote servers join the system, resulting in computing resources being much larger than those required by users. The server side needs to maintain resource's price at an optimal value to minimize the cost of task processing.

Furthermore, we can see that as the number of tasks increases, users require more computing resources, which necessitates the use of higher reward factors to attract more resource managers (RMs) to participate in the offloading system. However, reward factors cannot continue to increase indefinitely, as higher reward factors will lead to increased user costs. Competition among tasks leads to an increase in the computing resource price offered by the server, ultimately resulting in higher costs for each unit of task computation. Therefore, a balance must be struck between the reward factor and user costs to achieve optimal offloading decisions.

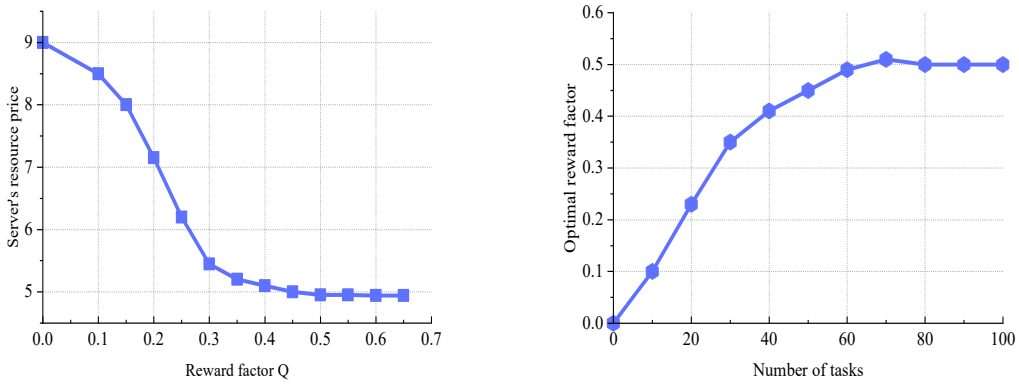

Fig.6. Correlation between optimal reward factor and number of tasks

As shown in Fig. 7, the game between users and servers becomes increasingly intense as the number of iterations increases, and the average resource price and service demand eventually reach an equilibrium point, demonstrating the effectiveness and convergence of the algorithm. Although the computing capacity of servers tends to reach saturation as the demand increases, leading to a gradual increase in resource prices, the competition among multiple users and the increasing number of iterations gradually leads to a stable state for both sides, resulting in the final convergence of the average resource price.

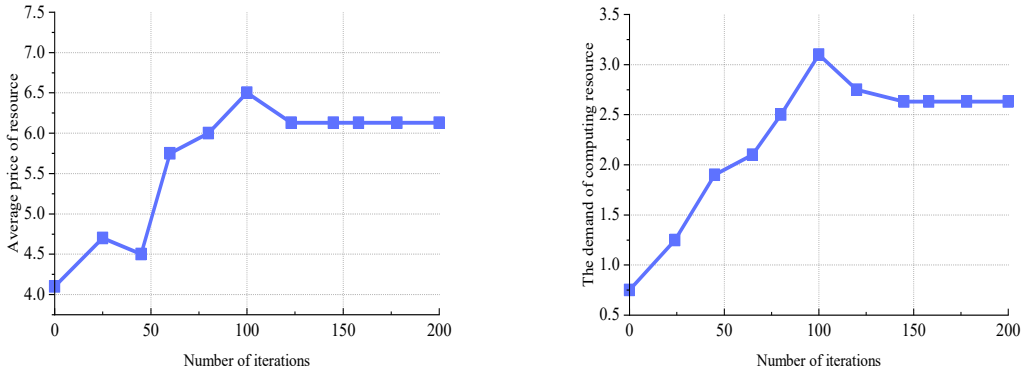

Fig. 7. Changes in average price of resource and demand of computing resource during iteration

The above simulation results validate the performance of the proposed algorithm under different settings. In the following section, the impact of the number of tasks and server's computing capacity on the server's utility function is analyzed. To verify the performance, we compare the proposed algorithm with three other algorithms:

CCOSGG: The proposed algorithm in this paper.

Joint Computation and Communication User Cooperation for MEC (JCCUC) [25]: In this scheme, a large amount of computation tasks is handled by using the surrounding idle small base stations when tasks are unevenly distributed. It is a classical cooperation offloading scheme.

Non-collaborative Offloading (N-CO) [26]: This solution uses a three-tier offloading scheme of UE, edge servers, and the CC to execute computation tasks, and it does not use collaborative offloading.

Random [27]: This solution randomly executes computation tasks at UE or transmits them to edge servers and the CC for execution.

By comparing the proposed algorithm with these three algorithms, we can evaluate the effectiveness of the proposed algorithm in terms of server utility.

In simulation experiments, the performance of different algorithms is measured by the utility function of the server.

Specifically, Fig. 8 show how changes in the number of tasks and the server's computing capacity affect the server's utility function. The results indicate that the server's utility function gradually increases as the number of tasks and the server's computing capacity increase. In CCOSGG, the server can earn higher revenue with the same number of tasks due to the additional profit bonus obtained by the remote edge server, thereby increasing its utility function. In other algorithms, servers with sufficient computing resources would lower the price of computing resources to attract users, leading to a decrease in overall profit. Therefore, servers prefer to retain idle computing resources, causing their utility function to reach the maximum value faster. While the utility function of CCOSGG grows relatively slowly, the maximum value it eventually reaches is higher.

Experimental results show that CCOSGG consistently outperforms other algorithms because it can set different prices for different tasks, enabling the server to fully utilize its computing resources and maximize its utility function. Furthermore, in CCOSGG, the relationship between the server's utility function and the user's utility function is inversely proportional, minimizing the overall system cost.

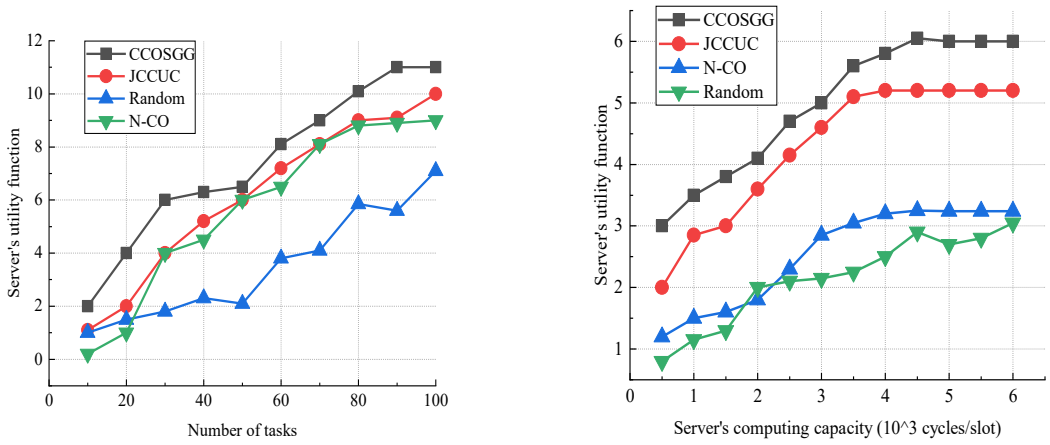

Fig. 8. Impact of the servers' computing capacity and number of tasks on the server's utility function

The proposed task clustering method is compared with classical horizontal and vertical clustering methods to evaluate its superiority. Horizontal clustering groups [28] sub-tasks with similar attributes into one cluster, whereas vertical clustering groups sub-tasks with dependencies into one cluster.

Analysis from Fig. 9 shows that vertical clustering has the highest average number of subtasks in a cluster, but the least waiting time between subtasks. This is attributed to the clustering of sub-tasks with dependencies and their execution as one sub-task, resulting in reduced waiting time. On the other hand, the horizontal clustering method has the least number of subtasks in a cluster, but the highest waiting time due to the presence of dependencies between clusters.

In the sub-task clustering method proposed in this paper, the number of sub-task clusters is appropriate, while the waiting time is relatively shorter in our approach. This is because our clustering algorithm is based on dependency, and we further divide clusters into smaller clusters with an appropriate number of sub-tasks, ensuring that each sub-tasks cluster can be processed on different servers while reducing waiting time. That is why the proposed sub-task clustering algorithm outperforms the horizontal and vertical clustering in terms of total system cost in Fig. 10.

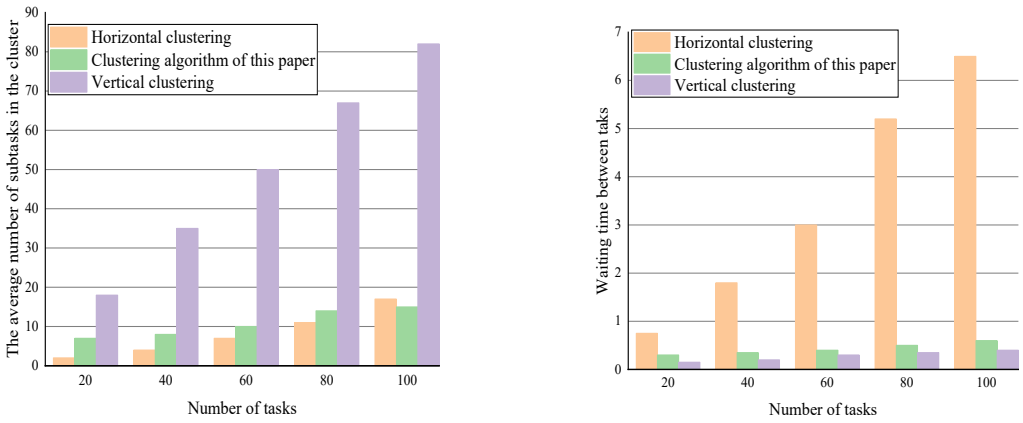

Fig. 9. Comparison of three clustering methods

In contrast, the horizontal clustering algorithm only groups sub-tasks based on similar attributes and fails to address the dependency problem, which can lead to long waiting times. While the vertical clustering algorithm resolves the dependency problem, clustering all sub-tasks with dependencies together results in a large number of tasks in the cluster, which may cause most sub-tasks to still be executed on cloud servers, increasing transmission delay.

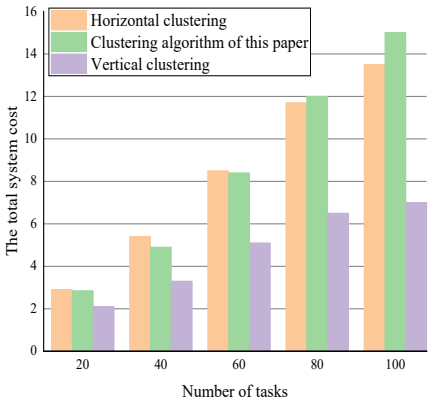

Fig. 10. Comparison of the three clustering methods in terms of total system cost

In summary, we have demonstrated that our task clustering method effectively reduces waiting time between sub-tasks, and each server can participate in collaborative offloading with reasonable incentive mechanisms, and our algorithm has excellent performance in reducing the overall system cost.

6 Conclusion

This paper addresses the problem of multi-edge collaborative offloading with sub-tasks dependencies. To reduce waiting time between sub-tasks, we propose a task clustering algorithm that clusters strongly correlated sub-tasks into one cluster using task reconstruction. Graph partition algorithm is used to divide large clusters into smaller ones, ensuring each server in the offloading system can execute them. To encourage remote edge servers to participate in collaborative offloading, we use Stackelberg game theory to introduce reasonable incentive mechanisms. Simulation results demonstrate that the CCOSGG algorithm has lower offloading costs for different numbers of users and is adaptable to various scenarios.

However, determining the leader in the Stackelberg game theory may raise fairness issues. Thus, we aim to introduce other factors, such as network bandwidth, to enhance the algorithm's fairness and efficiency. Additionally, exploring more sophisticated subtask clustering methods can improve the accuracy of the algorithm.

7 Acknowledgement

This work was supported by the Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX21_0732) and Jiangsu Provincial Key Research and Development Program (No. BE2020084-5).

References

[1] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, D. Sabella, On multi-access edge computing: a survey of the emerging 5g network edge cloud architecture and orchestration, IEEE Communications Surveys & Tutorials 19(3) (2017) 1657-1681.

[2] W.-Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, A. Ahmed, Edge computing: a survey, Future Generation Computer Systems 97(2019) 219-235.

[3] L. Fan, X. Liu, X. Li, D. Yuan, J. Xu, Graph4Edge: A Graph-based Computation Offloading Strategy for Mobile-Edge Workflow Applications, in: Proc. 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2020.

[4] J. Park, K. Chung, Collaborative Computation Offloading Scheme Based on Deep Reinforcement Learning, in: Proc. 2023 International Conference on Information Networking (ICOIN), 2023.

[5] F. Liu, J. Huang, X. Wang, Joint Task Offloading and Resource Allocation for Device-Edge-Cloud Collaboration with Subtask Dependencies, IEEE Transactions on Cloud Computing 11(3)(2023) 3027-3039.

[6] B. Li, K. Li, Y. Yuan, H. Ding, Deadline Constrained Computation Offloading Strategy in Cloud-Edge Collaborative Environments, in: Proc. 2022 IEEE 25th International Conference on Computer Supported Coofperative Work in Design (CSCWD), 2022.

[7] L. Wu, P. Sun, Z. Wang, Y. Li, Y. Yang, Computation Offloading in Multi-Cell Networks With Collaborative Edge-Cloud Computing: A Game Theoretic Approach, IEEE Transactions on Mobile Computing (2023) 1-14.

[8] C. Kai, H. Zhou, Y. Yi, W. Huang, Collaborative Cloud-Edge-End Task Offloading in Mobile-Edge Computing Networks With Limited Communication Capability, IEEE Transactions on Cognitive Communications and Networking, 7(2)(2021) 624-634.

[9] M.-D. Hossain, L.N.-T. Huynh, T. Sultana, T.D.T. Nguyen, J.H. Park, C.S. Hong, E.-N. Huh, Collaborative Task Offloading for Overloaded Mobile Edge Computing in Small-Cell Networks, in: Proc. 2020 International Conference on Information Networking (ICOIN), 2020.

[10] W. Hou, H. Wen, N. Zhang, W. Lei, X. Chen, A Device-Edge-Cloud Collaborative Framework for Hierarchical Computation Offloading, in: Proc. 2022 IEEE 19th International Conference on Mobile Ad Hoc and Smart Systems (MASS), 2022.

[11] Z. Song, R. Ma, Y. Xie, A collaborative task offloading strategy for mobile edge computing in internet of vehicles, in: Proc. 2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2021.

[12] C. Qian, G. Zhao, H. Luo, Game Theory based D2D Collaborative Offloading for Workflow Applications in Mobile Edge Computing, in: Proc. 2022 IEEE International Conference on Web Services (ICWS), 2022.

[13] L. Gu, M. Cui, L. Xu, X. Xu, Collaborative Offloading Method for Digital Twin Empowered Cloud Edge Computing on Internet of Vehicles, Tsinghua Science and Technology 28(3)(2023) 433-451.

[14] S. Tian, C. Chang, S. Long, S. Oh, Z. Li, J. Long, User Preference-Based Hierarchical Offloading for Collaborative Cloud-Edge Computing, IEEE Transactions on Services Computing 16(1)(2023) 684-697.

[15] R.-P. Lin, T.-Z. Xie, S. Luo, X.-N. Zhang, Y. Xiao, B. Moran, M. Zukerman, Energy-Efficient Computation Offloading in Collaborative Edge Computing, IEEE Internet of Things Journal 9(21)(2022) 21305-21322.

[16] M.-J. Feng, M. Krunz, W.-H. Zhang, Joint Task Partitioning and User Association for Latency Minimization in Mobile Edge Computing Networks, IEEE Transactions on Vehicular Technology 70(8)(2021) 8108-8121.

[17] M. Sheng, Y.-T. Wang, X.-J. Wang, J.-D. Li, Energy-Efficient Multiuser Partial Computation Offloading With Collaboration of Terminals, Radio Access Network, and Edge Server, IEEE Transactions on Communications 68(3) (2020) 1524-1537.

[18] M. Salmani, T.-N. Davidson, Energy-Optimal Multiple Access Computation Offloading: Signalling Structure and Efficient Communication Resource Allocation, IEEE Transactions on Signal Processing 68(2020) 1646-1661.

[19] C.-L. Ren, G.-A. Zhang, X.-H. Gu, Y. Li, Computing Offloading in Vehicular Edge Computing Networks: Full or Partial Offloading?, in: Proc. 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC), 2022.

[20] M. Guo, W. Wang, X. Huang, Y.-R. Chen, L. Zhang, L.-Y. Chen, Lyapunov-Based Partial Computation Offloading for Multiple Mobile Devices Enabled by Harvested Energy in MEC, IEEE Internet of Things Journal 9(11)(2022) 9025-9035.

[21] Z.-Y. Sun, Y.-J. Wang, G.-F. Chen, G.-W. Wu, Stackelberg Game-Based Task Offloading Strategy for Multi-Users, in: Proc. 2021 International Conference on Electronic Information Engineering and Computer Science (EIECS), 2021.

[22] H. Zhou, Z.-N. Wang, N. Cheng, D.-Z. Zeng, P.-Z Fan, Stackelberg-Game-Based Computation Offloading Method in Cloud–Edge Computing Networks, IEEE Internet of Things Journal 9(17)(2022) 16510-16520.

[23] X. Chen, L. Jiao, W.-Z. Li, X.-M. Fu, Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing, IEEE/ACM Transactions on Networking 24(5)(2016) 2795-2808.

[24] C. Sonmez, A. Ozgovde, C. Ersoy, EdgeCloudSim: An environment for performance evaluation of Edge Computing systems, in: Proc. 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), 2017.

[25] X.-W. Cao, F. Wang, J. Xu, R. Zhang, S.-G. Cui, Joint Computation and Communication Cooperation for Energy-

Efficient Mobile Edge Computing, IEEE Internet of Things Journal 6(3)(2019) 4188-4200.
[26] H. Badri, T. Bahreini, D. Grosu, K. Yang, Energy-Aware Application Placement in Mobile Edge Computing: A Stochastic Optimization Approach, IEEE Transactions on Parallel and Distributed Systems 31(4)(2020) 909-922.
[27] G. Cerutti, R. Prasad, A. Brutti, E. Farella, Compact Recurrent Neural Networks for Acoustic Event Detection on Low-Energy Low-Complexity Platforms, IEEE Journal of Selected Topics in Signal Processing 14(4)(2020) 654-664.
[28] S.M. Aaqib, An Efficient Cluster-Based Approach for Evaluating Vertical and Horizontal Scalability of Web Servers using Linear and Non-Linear Workloads, in: Proc. 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 2019.