Multi-User Offloading for Edge Computing Networks: A Dependency-Aware and Latency-Optimal Approach

Chang Shu, Zhiwei Zhao*, *Member, IEEE,* Yunpeng Han, Geyong Min*, *Member, IEEE,* Hancong Duan, *Member, IEEE,*

***Abstract*—Driven by the tremendous application demands, the Internet-of-Things (IoT) systems are expected to fulfill computation-intensive and latency-sensitive sensing and computational tasks, which pose a significant challenge for the IoT devices with limited computational ability and battery capacity. To address this problem, edge computing is a promising architecture where the IoT devices can offload their tasks to the edge servers. Current works on task offloading often overlook the unique task topologies and schedules from the IoT devices, leading to degraded performance and underutilization of the edge resources. In this paper, we investigate the problem of fine-grained task offloading in edge computing for low-power IoT systems. By explicitly considering 1) the topology/schedules of the IoT tasks, 2) the heterogeneous resources on edge servers and 3) the wireless interference in the multi-access edge networks, we propose a lightweight yet efficient offloading scheme for multi-user Edge systems, which offloads the most appropriate IoT tasks/subtasks to edge servers such that the expected execution time is minimized. To support the multi-user offloading, we also propose a distributed consensus algorithm for low-power IoT devices. We conduct extensive simulation experiments and the results show that the proposed offloading algorithms can effectively reduce the end-to-end task execution time and improve the resource utilization of the edge servers.**

I. Introduction

With the continuous development of microelectronic and communication technologies, massive novel applications have emerged, such as the smart access control based on face recognition [1], smart vehicular networks [2] and virtual reality [3]. These applications are often computation-intensive and latency-sensitive [4]–[6]. A natural obstacle for the IoT systems to accommodate such tasks is that IoT devices are often small in size (smartphones, wearables, etc.), which further leads to limited computation and communication abilities. To enhance the computation and communication capacity of IoT systems, Multi-Access Edge Computing (MEC) has recently been proposed as a promising technology to overcome this dilemma [7]–[9]. Specifically, edge computing for IoT is an ecosystem aiming to provide processing capabilities for the resource-constrained IoT devices by providing computation resources at the edge of IoT networks [10]. The IoT tasks are first uploaded to and then processed in the edge networks. After that, the results are returned to the corresponding IoT devices [11]. By offloading the tasks from resource constrained devices to the powerful edge servers, the overall execution delay and energy consumption can be reduced.

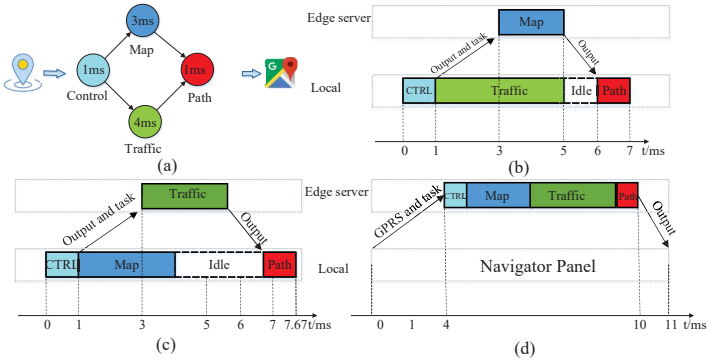

Fig. 1. DAG of Navigator app

Considering the increasing complexity of IoT applications [12] [13], a typical application task consists of a series of subtasks [1] [2], which are originally designed to enable multi-thread processing. The dependency among the subtasks can be often described by a directed acyclic graph (DAG). Fig. 1 shows the typical task topology of a vehicular navigation device. The circles denote subtasks of the application and the directed edges represent the intertask dependency. The values marked on each circle denotes the local execution time of the corresponding subtask. The navigation system in Fig. 1 runs as follows. The user first inputs a destination on the navigation devices, which activates the *Controller* module to check the current GPS position. Then the data flow passes through two parallel subtasks *Map* and *Traffic* to obtain all the optional paths and the traffic conditions along the direction to the destination. The last subtask, *Path*, interacts with the above two parallel subtasks to provide the best itinerary to the navigation panel.

Following most existing offloading works [14]–[17], these four subtasks are often packed and treated as an entire task. The upload decision does not distinguish the four different subtasks inside the navigation task. The reason is that they are considered as an "inseparable" unit due to the dependency [18]–[20].

C. Shu, Z. Zhao, Y. Han and H. Duan are with School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China. G. Min are with University of Exeter, Exeter, UK. Z. Zhao and G. Min are the corresponding authors.

TABLE I
THE DELAY COMPARISON

	Transmission	Execution	Overall
Uploading *Map*	3ms	6ms	7ms
Uploading *Traffic*	3ms	6.67ms	7.67ms
Uploading application	5ms	6ms	11ms

Considering the limited communication ability of the navigation devices, such offloading may lead to too much transmission delay and the overall execution time may not be reduced as shown in Fig. 1(d). As a result, the fine-grained offloading opportunities for subtasks are also wasted. We argue that in such scenarios, a more reasonable offloading could be to upload either *Map* (Fig. 1(b)) or *Traffic* (Fig. 1(c)) as these two subtasks can be executed at the same time. In this way, these two parallel subtasks are processed concurrently at the IoT devices and edge servers and the delay could be reduced. Table I compares the delay for uploading the entire task and uploading subtasks (It is worth noting that when uploading subtasks, the local execution and transmissions start at the same time and 2ms is overlapped).

We can see that for the low-powered scenarios with constrained resources, *we need to offload the most appropriate subtasks in order to exploit fine-grained offloading opportunities and minimize the overall delay*. However, it is a non-trivial task due to the following challenges.

1) **How to reveal the offloading opportunities from the task topologies.**
   From the example, we can see that according to the specific task topology, different subtasks have different potential benefits when offloaded (Fig. 1(b), and Fig. 1(c)). Therefore, before offloading specific subtasks, we need first to identify 1) which subtasks can be offloaded and 2) which combinations of subtasks provide more potential performance gains. For example in Fig. 1, although *Map*'s overall delay at the edge (including the transmission delay) exceeds its local execution delay (3ms), it can still be offloaded as it can be executed in parallel with *Traffic* and thus the overall delay for the entire task can be reduced. To address this challenge, we need to analyze the task topology to find out such offloading opportunities.
2) **How to coordinate subtasks uploading strategies among different users for the multi-user edge computing systems with heterogeneous servers.**
   The transmission time and execution time in the edge play an important role in the overall execution time for offloaded tasks, which can directly impact the offloading decisions. Therefore, we need to coordinate all the uploading strategies of different users to achieve a minimum average delay. However, the sub-task offloading problem showed in Fig. 1 will become much more challenging in multi-user, multi-subtask scenario as: 1) The transmission time is affected by the number of contending users, which is determined by the offloading decisions of other nearby users; 2) each user may offload different amount of task data, which also has an important impact on task scheduling in each server.
3) **How to determine the subtasks to be offloaded in a lightweight and consensus manner, with dynamically changing wireless and requests conditions.**
   With the above information, the next job is to determine which subtasks should be offloaded. The main challenge for the decision is that 1) how to evaluate the potential benefits of offloading a given set of subtasks and 2)the distributed consensus strategy is important in resource-constrained IoT devices. There is a high demanding for the coordination of subtasks to efficiently process data over the IoT devices or edge servers, while the processing power and storage capacity of wireless devices in IoT is rather restricted. Thus it is challenge for us to perform the decision process in a lightweight and consensus manner such that it does not incur much delay on the low-power IoT devices.

To overcome the above challenges, firstly we propose the EFO (Earliest Finish-time Offloading) algorithm for single-user MEC system to decide which subtasks need to be offloaded to achieve more performance gains. And then we further extend the EFO algorithm to multi-user systems with heterogeneous servers, aiming to coordinate the competition of communication and computation among multiple users. Moreover, in order to improve the efficiency of the offloading decisions, we design a distributed computation offloading algorithm that can achieve the Nash equilibrium. Finally, we conduct extensive simulations for performance evaluation. The results show that the proposed offloading algorithms can effectively reduce the end-to-end task execution time (by 81.6%) and improve the resource utilization of the edge servers.

The main contributions in this paper are summarized as follows:

1) We propose a potential evaluation scheme for subtasks, which considers not only the computation workload of the subtask but also the dependency among subtasks in the task DAGs. With this algorithm, we can find out the potential benefits for offloading specific subtasks and possible combinations.
2) Considering the competition among multiple users both in wireless communication and computation resources, we propose a coordinated EFO algorithm based on consensus algorithm, which coordinates the subtask uploading strategies based on the proposed EFO algorithm.
3) We implement the proposed algorithms and conduct simulation experiments for performance evaluation. Both centralized and distributed algorithms have been discussed and evaluated. The results show that the overall delay can be significantly reduced.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents the system model and problem formulation. Section 4 introduces the main design of the offloading strategy. Section 5 presents the evaluation results.

II. RELATED WORK

Recently, MEC (Multi-access edge computing) has received more and more attention, and various offloading policies have

been proposed [11], [21]–[25]. These policies can be classified into two categories: centralized and distributed.

For centralized offloading policies [15], [18], the MEC controller obtains all the information on the wireless channel quality and the computation requests from IoT devices. Based on the collected information, the controller arranges wireless channel and computation resources to each IoT device to avoid offloading collisions. In [18], the author studied resource allocation for multi-user MECO system, and the mobile users share a single edge server. They formulated this resource allocation strategy as a convex optimization problem, which aims to minimize the sum energy consumption. With the introduction of the multi-access network and ultra-dense network of 5G, Chen et al. in [15] proposed a task offloading policy for mobile edge computing in software-defined ultra-dense networks. They formulated the task offloading problem as a mixed integer non-linear program (which is NP-hard). Then they transformed this optimization problem into task placement sub-problem and resource allocation sub-problem. In the view of prior works, most of them assumed that the computation capacity of the edge server is infinite. Along a different line, in our work, we study the offloading scheme in multi-server, multi-user edge computing system and jointly consider the communication and resource scheduling at edge servers for offloaded tasks.

Another thrust of works targets distributed resource allocation for multi-user MEC system [8], [19], [20], [26]. In [8], [19] they formulate the distributed computation offloading decision-making problem among mobile users as a multi-user computation offloading game. And a distributed computation offloading algorithm is designed to achieve the Nash equilibrium. The upper bound of the convergence time is also derived. In [20], [26], targeting the multi-user and multi-server scenario of a small-cell network integrated with MEC, they formulate the problem as a distribute overhead minimization problem, which aims to minimize the overhead for users. Then they developed a potential game based offloading algorithm (PGOA) to save the offloading delay. Different from those distributed computation offloading strategies, we 1) consider the tasks as DAGs in order to identify the fine-grained offloading opportunities and 2) consider the competition among the subtasks from different users.

A closely related work to our is [14], the authors consider that the application consists of a series of sequential modules and then decide which module to be offloaded and how these modules should be executed (cloud computing or edge computing). An Iterative Heuristic MEC Resource Allocation (IHRA) algorithm is also proposed to dynamically make the offloading decisions. Compared to [14], our work considered more general and complex task dependencies in DAGs other than the sequential subtasks. Besides, we utilize the parallelism of subtasks in DAGs to minimize the overall delay. There has been extensive work on sub-task allocation and graph matching for datacenters [16], [17]. Our work has the following differences: 1) communications in MEC are via the wireless spectrum, thus experiencing more severe inference than communications in datacenters. 2) Most tasks in the datacenter scenario do not depend on any specific hardware. However, many tasks on IoT devices are restricted to be executed locally because they require specific sensors or modules on local devices. 3) The graph matching schemes in datacenters are computationally expensive for the resource-constrained IoT devices.

In this work, we propose a fine-grained offloading strategy to shorten application completion time, which jointly considers 1) the dependency of subtask in DAGs, 2) the offloading strategies in a multi-user MEC system with multiple heterogeneous servers, and 3) the scheduling of offloaded tasks on the edge servers. Compared to our prior work [27], we prove two theorems to confirm the problem is NP-hard and guide the design of the distributed and consensus algorithms. We also conduct more extensive experiments to study the algorithm performance under various network conditions.

III. System model

In this section, we introduce the details of the system model. The IoT system is composed of a code profiler, system profiler, and decision module [28]. The responsibility of code profiler is to determine which part of the code could be offloaded (depending on application type and code partitioned [29]). The system profiler monitors the parameters, such as wireless bandwidth, data size to be uploaded, and the energy spent by execution or transmission of the various subtask. Finally, the decision module determines whether the subtasks are to be offloaded or not. In our work, we mainly focus on the design of the decision module. The main objective of our decision module is to minimize the execution delay of the application while the energy consumption at the IoT devices is satisfied. Specifically, a decision on the task (context) offloading to the edge servers is made to check whether the offloading is profitable for IoT devices in terms of energy consumption and execution delay (Note that there is some non-offloadable code part that cannot be offloaded, e.g., user input, camera, or acquire position that needs to be executed at the IoT device).

We consider a set of $\mathcal{N}= [1, 2, ..i, ..N]$ IoT devices, where each user has an application-level task to be completed with the help of the edge servers. In the multi-access edge network or ultra-dense IoT network, there will be a variety of edge servers, defined by $\mathcal{S}= [1, 2, ..k, ..S]$. Considering that the IoT devices are equipped with only one radio, although several edge servers are available, only one of them can be selected by an IoT device during task execution. Consequently, for any IoT device i, it can choose at most two offloading strategies to finish its subtasks, i.e., computing locally or offloading to one of the edge servers. The computation-intensive application is partitioned into some interdependent subtasks, denoted by a set of $\mathcal{V}_i= [1, 2, ..j, ..V_i]$, $\forall i \in \mathcal{N}$. We utilize a directed acyclic task graph $G = (V, E)$ to describe the dependency relationships among these tasks as shown in Fig. 1, where V is the set of subtasks. We assume that each subtask is atomic and non-interruptible during execution. E is the set of directed edges characterizing dependency between the subtasks. For example, a directed $edge(m, n)$ implies that task n relies on the result task m. Each node label shows expected local execution time, the weight on each edge shows the amount

TABLE II
NOTATIONS

Notation	Description
$\mathcal{N}$	The set of IoT devices
$\mathcal{S}$	The set of edge servers
$\mathcal{V}_i$	The set of subtasks of devices i
$T_{i,j}$	$j-th$ subtask of the device i
$m_{i,j}$	The computation input data size of $T_{i,j}$
$c_{i,j}$	The required CPU cycles of $T_{i,j}$
$a_{i,j}$	The offloading strategy of $T_{i,j}$
a_i	The offloading scheme of device i
$\boldsymbol{a}$	The decision profile of all the devices
f_i^l	The local computation capability of device i
f^k	The computation capability of server k
$t_{i,j}^l$	The local execution time of subtask $T_{i,j}$
$e_{i,j}^l$	Energy consumption of $T_{i,j}$ for local task processing
$t_{i,j}^k$	The total latency of $T_{i,j}$ in server k
$w_{i,j}^k$	Execution time of $T_{i,j}$ in edge server k
$\overline{w_{i,j}}$	Average execution time of $T_{i,j}$
$e_{i,j}^k$	Energy consumption for $T_{i,j}$ processing in server k
$AST(j,k)$	Actual execution start time of subtask j in server k
$AFT(j,k)$	Actual execution finish time of subtask j in server k
$C_{j,j'}$	Transferring data from subtask j' to subtask j
$rank(j)$	The priority of subtask j

of data communication between two tasks. Considering the uncertainty of wireless channel and competition of the computation resource in each server, the latency minimization problem for offloading needs to address the communication and computation resources allocation at the same time. In the following, we continue to present the communication and computation models. For clear presentation, we summarize the notations used in the following formulation in Table II.

A. Communication Model

We consider a multi-user MEC system with multiple heterogeneous edge servers (containing heterogeneous resources). For any *j-th* computation subtask of IoT device i, i.e., $T_{i,j}$, it can be described with three terms, i.e., $T_{i,j}$=($m_{i,j}$, $c_{i,j}$, $a_{i,j}$). Here $m_{i,j}$ denotes the size of computation input data (e.g., the program codes and input parameters), $c_{i,j}$ denotes the required total CPU cycles to accomplish the task $T_{i,j}$. Let $a_{i,j} = \{0\} \cup S$ denote the offloading decision of the *j-th* computation subtask of IoT device i, where $a_{i,j}$=0 means the IoT device i chooses to execute the subtask $T_{i,j}$ locally with its own CPU, $a_{i,j}$=k, $\forall k \in \mathcal{S}$ means that $T_{i,j}$ will be offloaded to the *k-th* server for execution. It is worth mentioning that some subtasks are forced to execute locally. For example, in Fig. 1, as the subtask *controller* needs to collect the GPS information of the device, which makes it mandatory to execute the task locally. Note that if IoT device i decides to offload part of its subtask to server k, all the other subtasks at IoT device i to be offloaded can only select server k as the destination server. We adopt A_i=$\{a_{i,1}$,$a_{i,2}$,...$a_{i,v_i}\}$ to represent the offloading decision for device i. Let a_i be an indicator to denote where IoT device i execute its subtasks. a_i is obtained as:

$$a_i = \begin{cases} 0 & \text{if the application executed locally} \\ k & \text{at least one subtask offload to server } k \end{cases} \quad (1)$$

For a given decision profile $\mathbf{a}$=$(a_1, a_2, .., a_N)$ of all the devices, we can compute the uplink data rate between IoT device i and edge server k as follows:

$$r_{i,k}(\mathbf{a}) = Blog_2(1 + \frac{q_i g_{i,k}}{\varpi + \sum_{i' \in N: a_i = a_i'} q_{i'} g_{i',s}}) \quad (2)$$

Here B denotes the channel bandwidth, q_i is device $i's$ transmission power, and $g_{i,k}$ means channel gain for the link between device $i's$ and edge server k. ϖ denotes background noise power. $\sum_{i' \in N: a_i = a_i'} q_{i'} g_{i',s}$ denotes the wireless interference suffered from other IoT devices with the same offloading server. It is worth noting that IoT devices keep connected with the edge servers as long as there is at least one subtask needs to offload to it. Here our main focus is exploring the computation offloading problem under the wireless interference model, the other communication model and link features [30] are also potentially suitable for our work.

The computation subtask can be handled locally or offloaded to one of the edge servers for processing. Thus we will discuss the subtask processing time and energy consumption for both local and edge computing.

1) *Local Computing*: For local computing approach, let f_i^l be the computation capability of IoT device i. Then the local execution time of subtask $T_{i,j}$ is given by:

$$t_{i,j}^l = c_{i,j} / f_i^l \quad (3)$$

Next, we can obtain the energy consumption for local subtask as:

$$e_{i,j}^l = \delta_i^l c_{i,j} \quad (4)$$

where δ is the energy consumption per CPU cycle in IoT device i.

2) *Edge Computing*: In our scenario, the computation capability of different servers are heterogeneous. Let f^k denote the computation capability of server k. For the computation offloading, the subtask would incur extra overhead in terms of time and energy for transmitting the task data to the edge via wireless access. According to the communication model in Eq. (2). The total latency of subtask $T_{i,j}$ in edge server k is achieved by:

$$t_{i,j}^k = m_{i,j} / r_{i,k}((\mathbf{a})) + t_{i,j}^{k,Queue} + \omega_{i,j}^k \quad (5)$$

$$\omega_{i,j}^k = c_{i,j} / f^k \quad (6)$$

Here, $m_{i,j}/r_{i,k}((\mathbf{a}))$ is the transmission time invoked by the task offloading, and $\omega_{i,j}^k$ represents the execution time in edge server k. $t_{i,j}^{k,Queue}$ denotes the queuing time, which depends not only on the completion time of its predecessors but also on the task scheduling of subtasks offloaded by other users (Detailed discussion in Section 4). From Eq. (5) we can see that, we need to arrange the offloading strategy for each subtask of each device reasonably to minimize the delay of the whole system. On the other hand, from a distributed perspective, each IoT devices should make a rational decision to mitigate interference on the wireless channel and avoid long

queuing time in the edge server. The energy consumption of the IoT device is given by:

$$e_{i,j}^{k} = q_i m_{i,j} / r_{i,k}((\mathbf{a})) \tag{7}$$

B. Problem Formulation

In this paper, to jointly consider the subtask dependency of the application, task scheduling on the edge server, and the multi-user wireless interference, we formulate the following problem:

1) **Subtask placement problem**: To decide whether subtasks should be processed locally or offloaded to the edge servers based on the dependency among subtasks.
2) **Resource scheduling problem**: To order the different subtasks of different applications to reduce the overall delay for the global system.

Our aim is to minimize the average task duration of all the IoT applications. Before presenting the objective function, it is necessary to define the *AST* and *AFT* attribute. $AST(j,k)$ and $AFT(j,k)$ are the actual execution start time and the actual execution finish time of subtask j while offloading to edge server k. In order to compute the *AST* of task j, all immediate predecessor subtasks of j must have been scheduled, thus its value is computed recursively, starting from the entry task:

$$AST(j,k) = \max\{avail\{0 \cup [k]\}, \max_{j' \in pred(j)}(AFT(j') + C_{jj'})\} \tag{8}$$

where $pred(j)$ is the set of immediate predecessor subtasks of subtask *j*. $C_{j,j'}$ denotes the communication cost of $edge(j,j')$, which is incurred by transferring data from subtask j' to subtask j (more details can be found in Section 4). Thus inner max block in the equation returns the ready time when all data needed by subtask j has arrived at edge server k. After meeting the inter-dependency between tasks, let $avail[k] \cup 0$ be the earliest at which server k or local CPU is ready for the task execution, i.e, the earliest time when the CPU is idle after the ready time of subtask j. Thus it is an insertion-based policy which considers the possible insertion of a task on a server. By looping through all the servers with Eq. (8), we can get the earliest start time of subtask j. The actual execution finish time is defined as follow:

$$AFT(j') = \min\{\omega_{i,j}^{k'} + AST(j',k')\} \quad k' \in \{0\} \cup \{k\} \tag{9}$$

where $\omega_{i,j}^{k'}$ denotes the execution time in the edge server k or in the local CPU. After all subtasks in DAGs are scheduled, an application execution time will be calculated as the actual finish time of exit subtask n_{exit} (a task without any children is called an exit task):

$$AFT_i = AST(exit) + t_{i,v_i}^{L} \tag{10}$$

Without loss of generality, the final task is in charge of collecting the execution result. Hence, it is always executed on local devices. We utilize t_{i,v_i}^{L} to represent the execution time of the last subtask of IoT device i. After we obtain the actual finish time of each application, the task offloading problem can be formulated as follows:

$$\textbf{Opt}: Minimize_{a_{i,j} \in [S]^N}(\sum_{i=1}^{N} AFT_i)\ \forall i \in N\ \forall j \in \mathcal{V}_i \tag{11}$$

$$s.t \quad Cost_i \leq B \qquad \forall i \in N \tag{12}$$

$$Cost_i = \sum_{j=1}^{v_i} x_j c_{i,j} \delta_i + (1 - x_j) q_i \frac{m_{i,j}}{r_{i,k}(\mathbf{a})} \tag{13}$$

Our goal is to find a task assignment strategy for all subtasks from different IoT devices, to minimize the total latency and satisfies the cost constraints (defined in Eq. (12)). x_j denotes whether the subtask is executed locally ($x_j = 1$) or not ($x_j = 0$). This optimization problem is NP-hard (according to Theorem 1). In the following Sections, we will propose heuristic algorithms to solve our offloading problem efficiently.

Theorem 1. Problem **Opt** is NP-hard

Proof. We reduce our problem to the 0-1 knapsack problem. We simplify **Opt** to the problem, where there are only one IoT device and one edge server, then a binary partition is made on a serial task graph without considering data transmission between local an edge server. Thus the cost of transmission can be neglected. Our problem can be written as:

$$\textbf{Opt'}: Min_{x_i \in \{0,1\}} \sum_{j=1}^{V_i} (1 - x_i)\omega_{i,j}^{k} + x_i t_{i,j}^{L}) \tag{14}$$

$$s.t \sum_{j=1}^{v_i} x_j c_{i,j} \delta \leq B \tag{15}$$

Give N items with their value$\{v_1, ...v_N\}$, and weight $\{w_1, ...w_N\}$, the 0-1 knapsack problem can be formulated as:

$$\textbf{Knap}: max_{x_i \in \{0,1\}} \sum_{i=1}^{N} x_i v_i \tag{16}$$

$$s.t. \sum_{i=1}^{N} x_i w_i \leq B \tag{17}$$

Now **Knap** can be reduced to **Opt'** by the following assumption:

$$\omega_{i,k}^{k} = 0 \tag{18}$$

$$t_{i,j}^{L} = -v_i \tag{19}$$

$$c_{i,j}\delta = w_i \tag{20}$$

□

IV. SUBTASK OFFLOADING STRATEGY

In this section, we present an efficient task offloading scheme according to the optimization problem defined in Eq. (11). We first propose the EFO (Earliest-Finish time Offloading) algorithm for the single-user MEC system with only one edge server, which depends on not only the computation workload of the subtask but also the dependency relationship among subtasks in DAGs. We further extend the above EFO algorithm to the multi-user MEC systems with heterogeneous servers, aiming to coordinate the competition of communication and computation among multiple users. Moreover, to improve the efficiency of offloading decisions, we discuss on a distributed computation offloading algorithm to perform offloading decisions in a lightweight manner.

A. The EFO Algorithm

For the single-user MEC systems with only one server, the key issue is to schedule subtasks to make the latency as short as possible while meeting the task dependency requirement. There are two main problems to be resolved:

1) **Ordering subtasks by priorities:** In our work, the application always consists of a series of subtasks, which are often described by DAGs. How to order the subtasks by priorities is crucial. We cannot reverse the order of execution where there are task dependency relationships. Besides, changing the execution order of parallel subtasks can also impact the total latency. For example, if we execute the subtask *Traffic* before the subtask *Map*, the computation time of navigator will be increased to 7.67ms (Note that we still keep the capability proportion of edge server and local CPU). As a result, we need to set the priority of each subtask and minimize the overall delay accordingly.
2) **Processor selection problem:** After we obtain the priority of each subtask, we need to schedule each selected task on its "best" processor (local CPU or the edge server). As shown in Fig. 1(c), if we select the subtask *Traffic* as a prior one but inappropriately upload it the edge server and execution subtask *Map* locally. As a result, the computation time of navigator will also increase to 7.67ms.

As discussed before, an application of IoT device is represented by a directed acyclic graph, $G = (V, E)$. Let $Data$ be a matrix of communication data, where $data_{j',j}$ is the amount of data required to be transmitted from subtask j' to subtask j. And the communication cost of $edge(j', j)$ is defined by:

$$c_{j',j} = \begin{cases} 0 & \text{if } a_{i,j} = a_{i,j'} \\ data_{j',j}/r_{i,k}\textbf{(a)} & \text{otherwise} \end{cases} \tag{21}$$

when both subtask j and j' are scheduled on the same processor, $c_{j,j'}$ becomes zero since we assume that the intra-processor communication cost is negligible. The average communication cost of $edge(j, j')$ is defined by

$$\overline{c_{j,j'}} = data_{j',j}/(2r_{i,k}\textbf{(a)}) \tag{22}$$

The average execution cost of a subtask j is defined as

$$\overline{w_{i,j}^k} = (w_{i,j}^k + t_{i,j}^l)/2 \tag{23}$$

It should be explained that $\overline{w_{i,j}^k}$ denotes the average execution time of the subtask j. Parameters i and k are used in the next subsection, which denote the IoT device i and server k respectively in the multi-user and multi-server scenarios.

Subtask j is ordered in our algorithms by their scheduling priorities that are defined by the equation:

$$rank(j) = \overline{w_{i,j}^k} + \max_{j' \in Succ(j)} (\overline{c_{j,j'}} + rank(j')) \tag{24}$$

where $Succ(j)$ is the set of the immediate successor of task j. $rank(j)$ is the length of the critical path from task j to the exit task. The higher the value, the higher the priority. For example in Fig. 1, $rank(traffic)$ = (3+2.67)/2+0.5+1 equal to 4.33ms, and $rank(Map)$ =(3+2)/2+0.5+1 equal to 4ms, thus we should schedule the subtask *Traffic* at first. After ordering the priority of subtask, we then assign the subtask *Traffic* on its "best" processor, base on $AST(j)$ Eq. (8) and Eq. (9): *AST(traffic,local)*=max{1ms,1ms} +4ms and *AST(traffic, server k)* = max{0ms, (1+2)ms}+2.67ms. Therefore, the subtask should be executed locally.

The EFO algorithm is as follows:

Algorithm 1 The EFO (earliest finish-time offloading) Algorithm

1.**Input**:$G = (V, E)$, $Data_{i,j}$, $m_{i,j}$, $c_{i,j}$,f_i^l,f_i^k
2.**Output**: The server scheduling result, the total latency of the application
3. Sort the priority of all subtasks with Eq. 24 by non-increasing order
4. **while** there are unscheduled subtasks do
5. choose the subtask i with the highest priority;
6. compute the ($AFT(i)$) with Eq. 9 both in the edge server and local;
7 **if** $AFT_{local}(i) \leq AFT_{server}(i)$
8. assign subtask i to local
9. **else**
10. assign subtask i to edge server
11. **end if**;
12. **end while**

B. The EFO Algorithm in a multi-user scenario with multiple servers

In the next-generation network infrastructures (e.g., ultra-dense network, multi-access network), multiple heterogeneous edge servers can be deployed in the edge network to provide computation services to different users. As a result, in this subsection, we extend the EFO algorithm to the multi-user scenario with multiple servers. When different users offload their computing tasks, they have little information about the wireless channel conditions and the computation load of the edge servers, since they have no access to the offloading strategies of all other users. The strategy information includes which servers they choose, how many subtasks they will upload and the task scheduling in edge servers. Fortunately, a new network

paradigm SDWN (Software Defined Wireless Network) can achieve logically centralized control of the distributed IoT devices [31]. By utilizing the SDWN technology, each IoT device uploads its task-related DAGs and the local processing capacities to the controller at the beginning. Then the SDWN controller will determine where (different server) and when (task scheduling) to executed these subtasks which belong to different users.

The centralized EFO algorithm for multi-user and multi-server is shown in Algorithm 2.

Algorithm 2 The Centralize EFO Algorithm

1.**Input**: $\mathcal{N}$, $\mathcal{S}$, $G_i = (V_i, E_i), Data$, $m_{i,j}$, $c_{i,j}$,f_i^l,f_i^k
2.**Output**: The offloading strategy for different users, subtasks scheduling in each processor and the minimum latency of all the users
3.enumerate all optional offloading decision **AL**=$\{\boldsymbol{a}_1,\boldsymbol{a}_2...,\boldsymbol{a}_n\}$
4. **for all** $\boldsymbol{a}_i \in$ **AL** do
5. **for all** server $k \in \mathcal{S}$ do
6. Create the user group U_k, in which $a_i = k$
7. integrate G_i of the users in U_k to G_{u_k}
8. Update the transmission delay $r_{i,k}$***(a)***
9. Call EFO algorithm to obtain the subtask scheduling . in server k and local processor
10. **end for all**
11. Compute the overall computation delay for offloading . strategy $\boldsymbol{a}_i$
12. **end for all**
13.find the optimal offloading strategy $\boldsymbol{a}$ and its related subtask scheduling in each processor, which has the minimum computation delay for all the users

In the centralized EFO algorithm, we first enumerate all the optional offloading decisions. Let $\boldsymbol{a}_i$ denote one available offloading choice of all users. For each offloading scheme, we regard the task graphs of the users that offload to the same server as an integrated DAG by the union of the graph, thus we employ the EFO algorithm to obtain the overall computation delay of the scheme. Now we get to know the latency for each different offloading scheme. Finally, the offloading scheme with minimum latency is selected as the offloading strategy. According to Theorem 2, the convergence time of algorithm 2 will grows rapidly with the increase in the number of users and servers. We will then discuss and design a distributed computation offloading algorithm in order to improve the efficiency of the offloading decisions.

Theorem 2. The computational complexity of CEFO in Algorithm 2 is $O[(S+1)^N \cdot (n+e)]$, where S and N are the numbers of edge servers and IoT devices, respectively. n and e denote the number of a subtask and the edge of DAG.

Proof. The optional offloading strategies with S user and N edge server is equal to:

$$\binom{0}{N} \cdot S^0 + \binom{1}{N} \cdot S^1 + ...\binom{N}{N} \cdot S^N = \sum_{i=0}^{p} \binom{p}{N} \cdot S^p \quad (25)$$

where p denotes the number of users which decide to offload their tasks. According to Newton's binomial theorem:

$$\sum_{i=0}^{i} \binom{p}{N} \cdot S^p = (S+1)^N \quad (26)$$

Every optional offloading strategy needs to call the EFO algorithms to obtain the server scheduling result. The computational complexity of the first step in EFO is $O(n+e)$, if the DAG is store in Adjacency List. At the computational complexity of the second step in EFO is $O(n)$. Therefore the computational complexity of CEFO is equal to $[O(S+1)^N \cdot (n+e)]$ □

C. The distributed EFO algorithm

The centralized computational offloading strategy discussed in the above section would cause considerable overhead when the users and servers increase, thus it may be challenging to implement in ultra-dense IoT and edge networks. Even worse, it would possibly lead to system fails if the controller in a hardware failure. Moreover, as different IoT devices are usually owned by different vendors, it is hard to create the same standard for a variety of products. These factors inspire us to design a distributed computation offloading scheme. By the distributed methods, each IoT device can make the decision locally based on the information they collect (e.g., the channel conditions broadcasted by the edge servers). In our work, we adopt a game theoretic approach to coordinate competition among multiple users. Game theory is a powerful tool for devising distributed mechanisms with low complexity. In a game theory based offloading strategy among multi-user and multi-server, at each step a rational user react to other players' actions in the previous steps, and makes a locally optimal decision. After finite steps, all the users can self-organize into a mutual equilibrium state: the Nash equilibrium. In such a state no user can further reduce its delay by changing its strategy unilaterally. Let $\boldsymbol{a}_{-i}$ denote the computation offloading decisions of all the users except for user i. Given other user's strategies, user i would like to choose a locally optimal offloading decision for all its subtasks, by adopting which its overall delay can be minimized:

$$Minimize\ AFT_i(a_i, \boldsymbol{a}_{-i}) \quad (27)$$

In the initialization step of DEFO (distributed EFO) (**line 3**), each device executes all its subtasks locally. During every iteration, each device calls **Algorithm 1** to compute its *AFT* and offload tasks to different servers, based on the offloading result of the last iteration (**line 5-9**). In **line 10**, by utilizing the coordination and agreement algorithms in [32], the IoT devices vote on only one user with the greatest gain then the corresponding server broadcasts the offloading result of this iteration to all users(**line 11-16**). This process goes recursively. After finite iterations, the system finally achieves the Nash Equilibrium and we can obtain the offloading schemes for all IoT devices (**line 11-12**).

We try to explain our DEFO algorithm with an example shown in Fig. 2. There are three IoT devices and two edge servers in the network. The DAGs of the applications are

(a) The DAG of apps (b) The 2th iteration (c) The 3rd iteration (d) The equilibrium state

Fig. 2. Case Study of distributed EFO

Algorithm 3 The distributed EFO Algorithm

1.**Input**: $\mathcal{N}$, $\mathcal{S}$, $G_i = (V_i, E_i), Data$, $m_{i,j}$, $c_{i,j}$,f_i^l,f_i^k
2.**Output**: The offloading strategy for different users, subtasks scheduling in each processor and the minimum latency of all the users
3.**Initialize**: **a(0)**=$\{0,0,0,...0\}$ all the users execute locally
4.**for** each decision slot t do
5. **for all** $i \in$ **N** do
6. **for all** $k \in$ **S** do
7. call EFO algorithm to compute $AFT_{i,k}$, base on
. $r_{i,k}(\mathbf{a})$ and schedule result on server k at (t-1)
8. **end for**
9. **end for**
10.Find only one user i and the corresponding server k to achieve the maximum delay reduction
11. **if** Gain=1 **then**
12. **return a(t)**
13. **end if**
14.update **a(t)** and the task scheduling result on related server
15.update $r_{i,k}(\mathbf{a})$ based on **a(t)**
16.The related server broadcast its scheduling result and $r_{i,k}(\mathbf{a})$
17.**end for**

shown in Fig. 2(a). The initialization step of DEFO (**line 3**), each device executes all its subtasks locally. During the first iteration, every device calls the EFO algorithm to compute AFT according to a different target server for offloading (**line 4-8**). Note that at this iteration, every device assumes that there is no interference in the wireless channel and no other subtask executed on the edge server. And then the controller will choose $user_3$ to offload their subtask at $server_1$ as it achieves the maximum delay reduction compared to the local execution time(**line 10**). In (**line 14-16**), $user_3$ executes the EFO algorithm and offload $subtask_2$ and $subtask_4$ to $server_1$. Then the server $server_1$ will broadcast the scheduling results and its wireless channel rates $r_{i,k}(\mathbf{a})$ to all the IoT devices, which then facilitate the users to make further choices in the next iteration. In the second iteration, $user_1$ wins the updated opportunity as shown in the Fig. 2(b), and $server_2$ will broadcast the update information about the task scheduling result and wireless channel. In Fig. 2(c) user $user_2$ upload $subtask_2$ to $server_2$ according to the previous broadcast information. In final equilibrium state (Fig. 2(d)), since every user cannot further reduce its computation delay by changing its strategy unilaterally, $Gain = 1$ in **line 11**, the DEFO algorithm is then terminated.

We can see that with the above distributed EFO algorithm, each IoT device can implicitly take others' information into account and adjust its own offloading strategies accordingly. A possible drawback is that the convergence time may adds further overhead to the offloading system. We argue that this process incurs only an initial delay as normally the task DAGs do not change drastically for a given IoT system. Once the process is finished, each IoT device can perform their offloading decisions to reduce the overall task delay.

V. Evaluation

In this section, we evaluate the performance of the proposed offloading strategies. We study the performance improvements in different scenarios and the convergence time of the distributed EFO allocation algorithms in different conditions. We also compare the distributed EFO algorithm with the existing work, potential game based offloading algorithm (PGOA) in [20].

A. Experimental settings

We consider a multi-user MEC scenario in an ultra-dense IoT network (UDN) [33], where multiple IoT devices are randomly distributed in a $1km \times 1km$ area with some heterogeneous edge servers. The UDN network is emerging as one of core characteristics for 5G cellular networks [34], which introduces a new coverage environment where many edge servers are in the vicinity of a given IoT device. As a result, there are multiple choices of offloading destinations for each user to upload their subtasks. Our goal is to assign all subtasks from different IoT devices to the most appropriate edge servers to minimize the total latency. For simplicity, we make the following experimental settings.

- Each edge server/IoT device is equipped with one CPU.
- Each IoT device request one service, which consists of a series of inter-dependency subtasks to be executed with the help of no more than one edge server.
- Whether the subtasks should be offloaded depends on the total execution time of the entire application. More specifically, if the overall delay of offloaded application does not exceed the local completion time, IoT devices will choose to execute the application locally.
- In the distributed offloading strategy, IoT devices will continuously select the "best" edge server to offload their subtasks to achieve the minimum delay.

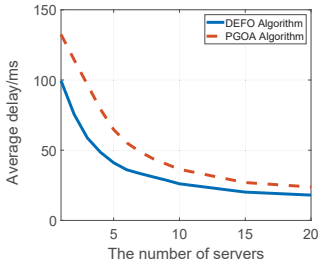

Fig. 3. Average delay for different network scales

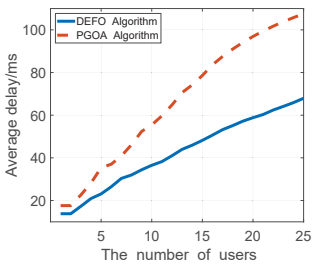

Fig. 4. Average delay for different user scales

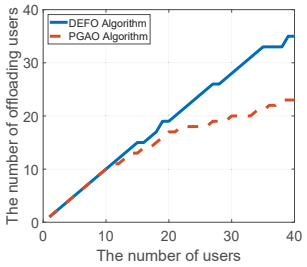

Fig. 5. Offloading users for different user scales

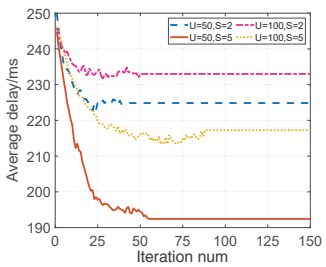

Fig. 6. The convergence of DEFO algorithm

- The DAG is generated according to the popular mainstream applications such as face recognition applications [35], pose recognition benchmarks [36] and video navigation application [37]. In order to verify the feasibility of our algorithms for various new applications, we also randomly generate some DAGs, where the CPU cycles required by each subtask varies from 20 *Megacycles* to 100 *Megacycles*.
- The transmission power q_i=100 *mWatts* [38], the background noise ϖ=-100 dBm [39] and the wireless channel bandwidth is 20 *MHZ*.
- We adopt from system measurements in [40] with the parameters of the IoT device and the cloud as follows: the CPU frequency of the edge server (500 MHZ) is ten times than that of IoT devices(5000 MHZ). The size of subtasks and its required CPU cycles are randomly generated between[500, 1500] *KB* and [0.2, 0.3] *GHZ*.

B. Convergence of the distributed EFO algorithm

As a game-theory based algorithm, the Nash equilibrium and the convergence of DEFO should be guaranteed. Fig. 6 illustrates the numerical results on the convergence behavior of DEFO with U=50, 100 S=2, 5 (U: the number of users S: the number of edge servers). During this group of repeated experiments, the DAG of the application is generated according to the face recognition applications [35], pose recognition benchmarks [36] and video navigation application [37]. These three applications are requested by the users following the uniform distribution. In Fig. 6, we observe that the distributed EFO can reach a Nash equilibrium after a finite number of iteration. We can also see that, 1) The convergence time will increase with the number of users. 2) The increased number of edge servers will contribute to reducing the average latency of users, as each user faces more opportunity to upload their subtasks, which comes with the increase of convergence time.

In edge computing systems, the applications usually have much more strict time requirements. From the above analysis, when the numbers of IoT devices and edge servers increase, the convergence performance will become a bottleneck factor for the whole system. To solve this problem, we propose two complementary policies:

1) The convergence time is manually controlled not to exceed a given threshold, in order to maintain the system stability. We control the convergence time via adjusting the iteration end time of each IoT device according to its delay requirement and execution time, i.e., before the convergence time of the whole system, the delay-sensitive IoT devices will stop finding a "better" offloading strategy, and will then begin to execute its subtasks. In case of a high number of IoT devices exist in the network, in each iteration ($line10$ in algorithm 1) the controller will comprehensively consider the delay requirement and the delay reduction by accepting the offloading request. In this way, the delay-sensitive applications have the priority to get the offloading opportunity.
2) The convergence time will also increase along with the number of edge servers, as the IoT devices face more opportunity to upload their subtasks. Thus we need to finish the iteration earlier and start to execute the application. Moreover, in each iteration of the algorithm, only one IoT device will be selected to update its offloading strategy. With the increasing number of edge servers, we allow selecting multiple users to update its offloading strategy concurrently (as long as these IoT devices select different offloading servers). Through this way, the convergence time can be reduced.

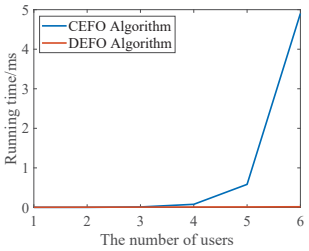

Fig. 7. Running time

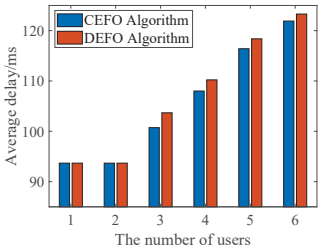

Fig. 8. Average delay

C. Comparison with the other schemes

In this section, we mainly compare DEFO with two different offloading strategies in edge computing scenarios:

1) The PGOA scheme is introduced in [20], they study a distributed computation offloading strategy for a multi-device and multi-server system and proposed the potential game-base offloading algorithm (PGOA) to solve this problem. However, they presented an offloading policy to decide whether an entire application should be offloaded in order to reduce execution time. Our work

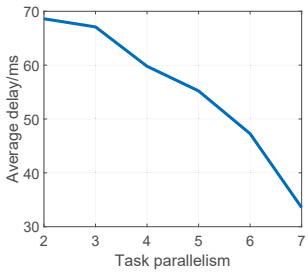

Fig. 9. Delay for different parallelism

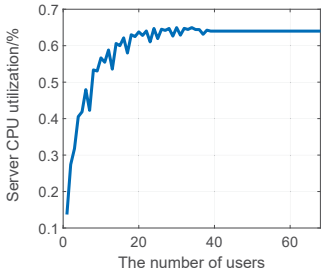

Fig. 10. Server utilization

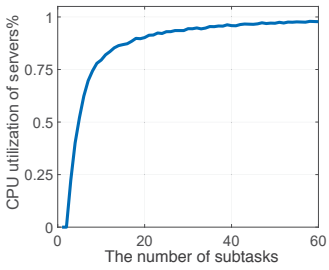

Fig. 11. Utilization for variety subtasks

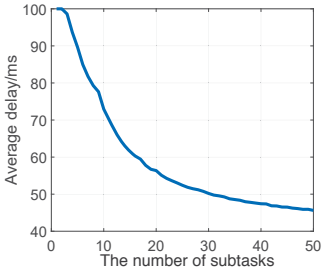

Fig. 12. Delay for variety subtasks

considers the subtasks inside the entire applications, and is expected to be able to identify more fine-grained offloading opportunities.

2) CEFO: As discussed in section 4, the SDN controller manages all the subtask offloading strategies and task scheduling in each edge server based on global information.

Fig. 3 shows the average task delay for different network scales in terms of edge servers. There are thirty users in the network. The number of the edge servers increases from two to twenty. It can be inferred that the average delay achieved by the distributed EFO algorithm is reduced compared to the PGOA algorithm. Fig. 4 shows the average delay for different user scales with the same edge network. The edge network consists of three edge servers. The number of users grows from 2 to 25 and DAGs of applications are randomly generated. We can see that as the number of users increases, the delay of EFO grows slower than PGOA. Fig. 5 shows the number of offloading users for different user scales. The offloading decision depends on the overall execution delay. From the perspective of mobile operators, the edge network should serve as many users as possible. From Fig. 5 we can see that as the number of users increases, it becomes difficult for the edge servers to serve all the users. The main reason for the improvements is that we deal with more fine-grained subtasks and thus have more chance to offload subtasks to the edge servers.

In order to verify the effectiveness of our proposed game-theory based EFO algorithm, we compare its running time and average delay with the centralized EFO algorithm. Fig. 7 and Fig. 8 illustrate the comparison results between them. The X-axis denotes the number of users (IoT devices). In consideration of CEFO's exponential computational complexity, the number of servers is set to 2, and the IoT devices from 1 to 6. As illustrated, although CEFO achieves a little higher average delay, it has a much higher computation efficiency and provides a workable solution in practice with a large number of IoT devices.

Currently, we do not consider the use of deep learning approaches as: 1) It will be extremely costly due to complex data models and task topologies. As there are thousands of different applications (the different relationship between subtasks) at the edge and their requirement change rapidly. The resource constrained IoT devices cannot afford the deep learning cost. 2) It is hard for deep learning to understand the causal relationship between inputs and outputs. We need to reveal the offloading opportunities from task topologies (DAG) through theoretically analyze. In this way, we have more opportunities to further optimize the offloading efficiency.

D. Impact of DAG properties on computation offloading

In this section, we first consider the task parallelism impact on the average delay. In Fig. 9, the X-axis denotes the parallelism of tasks, the degree of task parallelism is based on the number of tasks that can be executed simultaneously. As expected, when the parallelism of the task grows from 1 to 7, the average delay is declined to 75ms from 38ms. Fig. 10 shows the utilization of the server for different user scales. The DAG of each task consists of ten subtasks. We can observe that, when the number of users increases to 30, the server resource utilization no longer increase (up to 62%). This simulation result can be used to guide the edge server deployment for a given number of users. Through our simulation, we propose an effective method to further improve resource utilization shown in Fig. 11. There are 30 users in the network and the number of subtasks for each user increases from 2 to 60. In particular, although the number of subtask increases, the computing resources required by each user remain unchanged. It can also be inferred that through partitioning the application into more subtasks (in a more fine-granularity way), the utilization of server can be further improved. More importantly, the average delay can be reduced through fine-grained task partition as shown in Fig. 12.

Through analyzing the characteristics of the results shown in Figure 9-12, we further elucidate the target applications in our system model: 1) The parallelism among the subtasks in the application is larger than one. Otherwise, our offloading strategy is equal to the IHRA in [14], in which the authors consider that the application consists of a series of sequential subtasks. 2) The higher parallelism of application can lead to reduced execution time. In our further work, we can split the application into more parallel modules according to [41]. Our offloading strategy can speed up the execution of some parallel machine learning algorithms by utilizing their high degree of parallelism [42]–[44]. For example, in [42], the authors developed GraphLab, which improves MapReduce by compactly expressing asynchronous iterative algorithms with sparse computational dependencies and achieving a high degree of parallel performance.

VI. Conclusion

In this paper, we provide a fine-grained offloading strategy in edge computing for IoT systems. By jointly considering the dependency among subtasks and the contention among multiple edge users, we propose a novel offloading scheme to reduce the overall completion time of the IoT applications. We also discuss the distributed algorithms for resource constrained IoT devices and its convergence. We conduct simulation experiments and the results show that the proposed work can effectively reduce the overall application delay compared to the existing works.

Acknowledgement

This work was supported by the National Key Research and Development Program of China (2017YFB1400102), the National Natural Science Foundation of China (No. 61602095 and No. 61972074), the China Postdoctoral Science Foundation (No. 2018M640909), the National Postdoctoral Foundation for Innovative Talents (No. BX201700046), the Quacomm Research Funds (Tsinghua) and the European FP7 under Grant No: PIRSES-GA-2013-612652. Zhiwei Zhao and Geyong Min are the coresponding authors.

References

[1] L. Aceto, A. Morichetta, and F. Tiezzi, "Decision support for mobile cloud computing applications via model checking," in *Proceedings of 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE, 2015, pp. 199–204.

[2] W. Sun, J. Liu, and H. Zhang, "When smart wearables meet intelligent vehicles: Challenges and future directions," *IEEE wireless communications*, vol. 24, no. 3, pp. 58–65, 2017.

[3] M. Erol-Kantarci and S. Sukhmani, "Caching and computing at the edge for mobile augmented reality and virtual reality (ar/vr) in 5g," in *Ad Hoc Networks*. Springer, 2018, pp. 169–177.

[4] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 workshop on mobile big data*. ACM, 2015, pp. 37–42.

[5] 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.

[6] H. Yin, X. Zhang, H. H. Liu, Y. Luo, C. Tian, S. Zhao, and F. Li, "Edge provisioning with flexible server placement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1031–1045, 2016.

[7] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[8] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 4, pp. 3435–3447, 2017.

[9] Q. Fan, H. Yin, L. Jiao, Y. Lv, H. Huang, and X. Zhang, "Towards optimal request mapping and response routing for content delivery networks," *IEEE Transactions on Services Computing*, 2018.

[10] Z. Zhao, G. Min, W. Gao, Y. Wu, H. Duan, and Q. Ni, "Deploying edge computing nodes for large-scale iot: A diversity aware approach," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3606–3614, 2018.

[11] X. Zhang, H. Huang, H. Yin, D. O. Wu, G. Min, and Z. Ma, "Resource provisioning in the edge for iot applications with multi-level services," *IEEE Internet of Things Journal*, 2018.

[12] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[13] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[14] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet of Things Journal*, 2018.

[15] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[16] A. C. Zhou, S. Ibrahim, and B. He, "On achieving efficient data transfer for graph processing in geo-distributed datacenters," in *Proceedings of 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1397–1407.

[17] L. Chen, S. Liu, B. Li, and B. Li, "Scheduling jobs across geo-distributed datacenters with max-min fairness," *IEEE Transactions on Network Science and Engineering*, 2018.

[18] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[19] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[20] L. Yang, H. Zhang, X. Li, H. Ji, and V. Leung, "A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 6, pp. 2762–2773, 2018.

[21] Z. Yu, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, and N. Georgalas, "Federated learning based proactive content caching in edge computing," in *Proceedings of 2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[22] E. N. Pencheva, I. I. Atanasov, P. K. Penchev, and V. G. Trifonov, "Web service interfaces for intra-cell terminal activity," in *Proceedings of the 13th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS)*. IEEE, 2017, pp. 124–127.

[23] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[24] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: A taxonomy," in *Proceedings of the Sixth International Conference on Advances in Future Internet*. Citeseer, 2014, pp. 48–55.

[25] H. El-Sayed, S. Sankar, M. Prasad, D. Puthal, A. Gupta, M. Mohanty, and C.-T. Lin, "Edge of things: The big picture on the integration of edge, iot and the cloud in a distributed computing environment," *IEEE Access*, vol. 6, pp. 1706–1717, 2018.

[26] X. Ma, C. Lin, X. Xiang, and C. Chen, "Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing," in *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. ACM, 2015, pp. 271–278.

[27] C. Shu, Z. Zhao, Y. Han, and G. Min, "Dependency-aware and latency-optimal computation offloading for multi-user edge computing networks," in *Proceedings of IEEE SECON 2019*. ACM, 2019.

[28] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: from concept to practice and beyond," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 80–88, 2015.

[29] M. Deng, H. Tian, and B. Fan, "Fine-granularity based application offloading policy in cloud-enhanced small cell networks," in *2016 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2016, pp. 638–643.

[30] Z. Zhao, W. Dong, J. Bu, T. Gu, and G. Min, "Accurate and generic sender selection for bulk data dissemination in low-power wireless networks," *IEEE/ACM Transactions on Networking (ToN)*, vol. 25, no. 2, pp. 948–959, 2017.

[31] T. Luo, H.-P. Tan, and T. Q. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *IEEE Communications letters*, vol. 16, no. 11, pp. 1896–1899, 2012.

[32] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. pearson education, 2005.

[33] M. Kamel, W. Hamouda, and A. Youssef, "Ultra-dense networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2522–2545, 2016.

[34] X. Ge, S. Tu, G. Mao, C.-X. Wang, and T. Han, "5g ultra-dense cellular networks," *arXiv preprint arXiv:1512.03143*, 2015.

[35] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Proceedings of 2012 IEEE symposium on computers and communications (ISCC)*. IEEE, 2012, pp. 000 059–000 066.

[36] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile

devices," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 43–56.
[37] S. E. Mahmoodi, R. Uma, and K. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Transactions on Cloud Computing*, 2016.
[38] E. Access, "Further advancements for e-utra physical layer aspects," *3GPP Technical Specification TR*, vol. 36, p. V2, 2010.
[39] D. Wu, J. Wang, R. Q. Hu, Y. Cai, and L. Zhou, "Energy-efficient resource sharing for mobile device-to-device multimedia communications," *IEEE Transactions on Vehicular Technology*, vol. 63, no. 5, pp. 2093–2103, 2014.
[40] F. Lobillo, Z. Becvar, M. A. Puente, P. Mach, F. L. Presti, F. Gambetti, M. Goldhamer, J. Vidal, A. K. Widiawan, and E. Calvanesse, "An architecture for mobile computation offloading on cloud-enabled lte small cells," in *2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2014, pp. 1–6.
[41] C.-C. Kao, "Performance-oriented partitioning for task scheduling of parallel reconfigurable architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 858–867, 2014.
[42] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.
[43] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.
[44] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.