Efficient Task Offloading with Dependency Guarantees in Ultra-Dense Edge Networks

Yunpeng Han^1, Zhiwei Zhao^{1,*}, Jiwei Mo^1, Chang Shu^1 and Geyong Min^{2,*}
^{1}University of Electronic Science and Technology of China, China; ^{2}University of Exeter, UK;
{yunpeng,zhiwei,jiwei,shuchang}@mobinets.org; g.min@exeter.ac.uk

***Abstract*—The last decade has witnessed the rapid development of Internet of Things (IoT). The IoT applications are becoming more and more computation-intensive and latency-sensitive, which pose severe challenges for the resource-constrained IoT devices. To empower the computational ability of the IoT systems, edge computing emerges as a promising approach which allows the resource-constrained devices to offload their tasks to edge servers. A major challenge, which has been overlooked by most existing works on task offloading, is the dependencies among tasks and subtasks, which can have a significant impact on the offloading decisions. Besides, the existing works often consider offloading tasks to specific edge servers, which may underutilize the edge resources in the ultra-dense edge networks. In this paper, we investigate the problem of dependency-aware task offloading in ultra-dense edge networks. Specifically, we explicitly analyze the task dependency as directed acyclic graphs (DAGs) and establish full parallelism between edge servers and IoT devices. We further formulate task offloading as a joint optimization problem for minimizing both task latency and energy consumption. We prove the problem is NP-hard and propose a heuristic algorithm, which guarantees the dependency among subtasks and improves the task efficiency. Simulation experiments demonstrate that the proposed work can effectively reduce the task latency in ultra-dense edge networks.**

I. Introduction

With the development of microelectronic technologies and embedded computing in recent years [1], [2], massive IoT (Internet-of-Things) applications have emerged, e.g., face recognition, self-driving, smart healthcare and virtual reality (VR) [3] [4]. Many of these applications need intensive computation resources and strict latency guarantees, which result in high energy consumption on the end devices. Unfortunately, most IoT devices have limited computation and battery capacity, which can hardly support such complex applications. Rather than developing lightweight algorithms, one promising direction is Mobile Edge Computing (MEC) [5] [6]. Different from traditional cloud computing, servers in MEC are deployed geographically closer to mobile users, and thus reducing considerable communication delay [7]. To fully exploit this advantage, ultra-dense edge network is a promising direction that has attracted much research attention [8] [9]. By deploying a large number of connected edge servers for the target mobile network, mobile users (IoT devices) will gain pervasive and seamless computation and communication support from the edge servers [10]–[12]. In ultra-dense edge networks, the edge servers can communicate with each other with high-speed connections. Different subtasks of one application on a mobile device can be offloaded to different edge servers [13]. As a result, the computation-intensive and latency-critical applications from IoT users can be well supported.

Task offloading is one of the key technologies in the edge computing paradigm, which decides when and how a task should be offloaded to the edge servers. Given a complex computation task, offloading some subtasks to the edge servers can possibly enable parallel execution at both edge servers and end devices. This parallelism, however, can be severely impacted by the dependency of the subtasks. For example, if two subtasks are independent with each other, they can be executed in parallel. If one subtask depends on the output of another subtask, then these two tasks must be executed sequentially. In this case, although the energy consumption at end devices can be reduced, the overall delay may not be reduced as the offloaded subtasks incur round-trip communication delay between end devices and edge servers.

There are some pioneer works on the task offloading for edge computing. For example, Yang $et\ al.$ [14] proposed Potential Game based Offloading Algorithm (PGOA) based on game theory to perform distributed computation offloading; Chen $et\ al.$ [15] proposed Software Defined Task Offloading (SDTO) to minimize offloading delay while saving the battery life of devices. However, these works may underutilize the edge resources and add extra delay for the task executions, because most of them overlook the subtle dependencies among the subtasks.

There are also some existing works that consider task dependencies. Ning $et\ al.$ [17] consider task dependencies as serial task graph and propose a heuristic algorithm to coordinate resource competition among multiple users. Kao $et\ al.$ [13] consider task dependencies as tree-structured task graph and propose a lightweight offloading strategy with performance guarantee. These works are restricted to specific task scenarios and cannot support more general task offloading in edge computing. Different from the above works, we view the tasks as directed acyclic graphs (DAGs) in this paper, which are more general in representing various task topologies. Some existing studies have shown that DAG can generally support different kinds of computational tasks [18]. Based on DAG-tasks, we propose a joint optimization model for reducing task latency and energy consumption and then propose a heuristic algorithm, which considers both dependencies and priorities of subtasks. The task dependency is guaranteed and the execution efficiency between edge servers and end devices is improved. With simulation experiments, we demonstrate that the efficiency of task execution is largely improved compared to the existing works.

The major contributions include:

- We investigate the problem of task offloading with general dependencies among subtasks and formalize the task dependency as directed acyclic graphs (DAGs).
- We formulate a DAG-task offloading model to jointly optimize latency and energy consumption and design a heuristic algorithm, which considers both dependencies and priorities of the subtasks. The task dependency is guaranteed and the execution efficiency is improved.
- We conduct simulation experiments to study the performance of the proposed algorithm. The results show that compared to the existing works, the proposed algorithm significantly reduces execution delay and improves energy efficiency.

The rest of paper is organized as follows. Section II reviews related works. The system model is described in Section III. We present the heuristic priority-based DAG-task offloading algorithm in Section IV. The evaluation results are given in Section V. Finally, we conclude the paper in Section VI.

II. Related Works

Mobile Edge Computing (MEC) has attracted wide attention from both academia and industry, leading to a number of recent research works. We summarize the existing works focusing on three important aspects: offloading scenario, task model and offloading strategy.

A. Offloading scenario

The MEC offloading scenarios can be divided into three categories: (1) single-server with single user, (2) single-server with multiple users and (3) multi-server with multiple users. For the first scenario, Mao $et\ al.$ [19] investigate the one-server MEC system with one energy harvesting device. Based on Lyapunov optimization, the authors propose a low-complexity online algorithm. For the second scenario, Chen $et\ al.$ [20] study the multi-user computation offloading problem in single-server MEC system. They prove that the optimization problem is NP-hard, then a distributed offloading scheme is proposed based on game theory and this scheme can achieve Nash equilibrium after finite iterations. Guo $et\ al.$ [21] introduce hybrid fiber-wireless networks as a supplement to the edge server and present two schemes for the cloud-MEC computation offloading. For the last and also most complex scenario, Kao $et\ al.$ [13] study the multi-user computation offloading in the multi-server networks. They propose $Hermes$ based on dynamic programming, then prove the time complexity and performance guarantee of their algorithm. However, except for [13], all the works above did not consider the coordination among edge servers, which could yield significant benefits if properly utilized.

In this paper, we focus on the multi-server and multi-user MEC scenario, where the cooperation among edge servers is also considered.

B. Task model

The task model in current literature can be summarized into two categories: the atomic task model and the divisible task model. The atomic task model views the tasks as indivisible entities, i.e., a user's task must be executed as a whole either on local devices or on edge servers. [19], [22] and [23] and adopt this kind of task model. When it comes to divisible task model, most existing works focus on the serial task graph, where the subtasks must be executed one by one. [17] and [24] are two works adopting the serial task model. [13] and [25] investigate the tree-structured task offloading. [13] focuses on the tree-structured task graph, and the proposed heuristic algorithm optimizes the execution latency with the limit of energy consumption based on dynamic programming. [25] proposes *Clonecloud*, a system that automatically offloads applications to cloud. This system builds profile tree for each task, then efficient offloading strategy is selected based on an application analyzer. As for the general task dependency model, there is even less literature. [26] models the tasks as general dependency graph but only uses ILP solver for brute-force solving.

Different from the existing works, we view the tasks as DAGs, which is capable of representing more general tasks.

C. Offloading strategy

As mentioned above, there are already some works that consider task dependency. In view of the complexity of task dependency, how to deal with it is the key of offloading strategies. [13] explores the inherent recursive properties of tree-structured task graph and utilizes dynamic programming to search the optimal offloading strategy. The fly in the ointment is that their heuristic algorithm may take a long time when task graph is large. Shu $et\ al.$ [27] focus on the application offloading with general subtask dependency. Based on game theory, they develop an adaptive distributed offloading strategy that achieves Nash Equilibrium after finite iterations. However, this algorithm could only obtain limited performance due to sparse or even outdated global information. What's more, this work does not consider energy consumption, which is a key limiting factor for edge devices.

Different from the above works, in this paper, we propose a centralized algorithm based on the priorities of subtasks. This algorithm jointly optimizes the execution latency and energy consumption, while retaining low time complexity.

III. System Model

A. Overview

In our offloading problem, there are $\mathcal{N}= \{1, 2, ..i, ..N\}$ mobile devices and $\mathcal{M}= \{1, 2, ..m, ..M\}$ edge servers. Each device has one application to be completed. The execution procedure of application on device i can be represented by a DAG $G_i = (V_i, E_i)$. The node set V_i denotes a series of subtasks of application on device i. Meanwhile, some of the subtasks have dependency relationships, represented by the directed edge set E_i. An example of such a DAG is shown in Fig. 1, where the weights marked on nodes represent computation overhead of corresponding subtasks while the weights on the directed edges denote the amount of communication overhead between two subtasks. For edge networks, $\mathcal{M}$ edge servers are connected to each other by high-speed fiber networks. Any subtask can be offloaded to any edge server in the network or executed locally. It's worth highlighting that different subtasks on one device can also be offloaded to different edge servers.

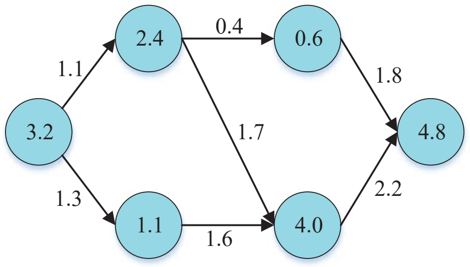

Fig. 1. An example of a DAG-task

Since we consider the offloading problems of DAG-tasks, an crucial observation is that we cannot reverse the execution order of two subtasks which have dependency relationships. In the following part, we present our DAG-task offloading model. Our target is to optimize the average execution latency under the energy-consumption limit. We first propose the computation and energy-consumption model in this section. Then, we formalize the optimization problem in Section IV.

B. The Computation Model

We adopt a non-preemptive task scheduling strategy, where the execution of one subtask cannot be interrupted as long as it is not finished. The execution latency of one application is accounted as the duration from the start of the initial subtask to the finish of the last subtask. We first investigate the finish time of the subtasks.

Since one subtask is executed either on edge server or locally, we can formulate actual earliest finish time of subtask k of application G_i on device i as follows:

$$EFT_A(i,k) = \min_{m\in\{\mathcal{M}\cup\{-i\}\}} \{EST_A(i,k,m) + T^m_{i,k}\} \quad (1)$$

Note that $m = -i$ denotes local device i. $T^m_{i,k}$ is the execution time of subtask k on corresponding edge server (or local device) m. $EST_A(i,k,m)$ is the actual earliest start time of subtask k on device i while offloading to m. Obviously, one subtask cannot start until all of its predecessor subtasks have completed. Also, this subtask cannot be scheduled if the processor is performing other tasks. Based on this observation, we can obtain the EST_A as follows:

$$EST_A(i,k,m) = \max\{avail\{i,k,m\}, EST_T(i,k,m)\} \quad (2)$$

$EST_T(i,k,m)$ is the theoretical earliest start time of subtask k on edge server (or local device) m, i.e., the time when all the input data needed for subtask k has been generated. $avail\{i,k,m\}$ is the earliest time at which edge server (or local device) m is ready for task execution, i.e., the earliest time when edge server m is idle and the idle timeslot is long enough to insert subtask k for execution. The reason is that we adopt an insertion-based scheduling policy. Note that $avail\{i,k,m\}$ will never be less than $EST_T(i,k,m)$ otherwise the task dependency cannot be guaranteed. $EST_T(i,k,m)$ can be recursively defined as follows:

$$EST_T(i,k,m) = \max_{k'\in pred(k)} (EFT_A(i,k') + C^{m'm}_{k'k}) \quad (3)$$

$pred(k)$ is the set of immediate predecessor subtasks of subtask k. $C^{m'm}_{k'k}$ denotes the data exchange time between subtask k' and k when k' is executed on m' and k is executed on m. It can be formulated as follows:

$$C^{m'm}_{k'k} = \frac{data_{k'k}}{B_{m'm}} \quad (4)$$

$data_{k'k}$ is the amount of data exchange between subtask k' and k, $B_{m'm}$ is the bandwidth between edge server m' and m. If $m = m'$, k and k' are executed at the same place, $C^{m'm}_{k'k} = 0$. We adopt an orthogonal frequency-division multiple access (OFDMA) scheme, thus there is no competition for wireless resources.

We can see that the EFT_A in Eq.(1) is also a recursive definition, the exit condition of this recursion is:

$$EST_T(i,1,m) = 0, \ \ \forall m \in \{\mathcal{M}\cup\{-i\}\}, \forall i \in \mathcal{N} \quad (5)$$

Eq.(5) denotes that the theoretical earliest start time of the initial subtask (serial number is 1) on each device is 0.

Obviously, $EFT_A(i,|V_i|)$ denotes the actual earliest finish time of the last subtask (serial number is $|V_i|$) on device i, which is also the minimum execution latency of the application G_i on device i.

C. The Energy-consumption Model

We then consider the energy consumption of mobile devices. We first set an offloading decision pofile $\boldsymbol{A}^k_i = \{a^k_i, i\in\mathcal{N}, k\in V_i\}$, where $a^k_i = -i$ denotes subtask k on device i is determined to be executed locally and $a^k_i = m, m\in\mathcal{M}$ denotes that k is offloaded to edge server m for execution. The energy consumption of mobile devices consists of two parts, computation consumption and transmission consumption. Thus the energy consumption can be formulated as follows:

$$Cost = Cost_c + Cost_t \quad (6)$$

$Cost_c$ is the computation-energy consumption of mobile devices, while $Cost_t$ is the transmission-energy consumption of mobile devices.

The $Cost_c$ depends on the specific subtasks executed locally. It is calculated as:

$$Cost_c = \sum_{i=1}^{|\mathcal{N}|}\sum_{k=1}^{|V_i|} I_{\{a^k_i=-i\}}\chi_i T^{-i}_{i,k} \quad (7)$$

I_X is a binary indicator function, $I_X = 1$ if and only if condition X is true, otherwise $I_X = 0$. χ_i is the computation-energy consumption of device i per unit time.

The transmission consumption comes from two parts: uploading tasks and data exchange among subtasks. It can be formulated as follows:

$$Cost_t = \sum_{i=1}^{|\mathcal{N}|}\sum_{k=1}^{|V_i|}\zeta_i(I_{a_i^k>0}\frac{Size_{i,k}}{B_i^{a_i^k}} + \sum_{k'\in pred(k)} I_{a_i^{k'}a_i^k<0}C_{k'k}^{m'm}) \tag{8}$$

ζ_i is the transmission power of device i. $Size_{i,k}$ is the bitwise size of subtask k. $B_i^{a_i^k}$ denotes the bandwidth between device i and edge server a_i^k.

IV. Efficient DAG-task offloading Scheme

In this section, we first present the optimization problem. After that, we propose a heuristic offloading algorithm to perform DAG-task offloading efficiently.

A. Optimization Problem

From Eq.(2), we can discover that the scheduling order of subtasks also affects the overall execution latency. As a result, we need to model the scheduling sequence. We define $\boldsymbol{B}_i^k = \{b_i^k, i \in \mathcal{N}, k \in V_i\}$ as the scheduling sequence of subtasks. Given $a_i^k = a_{i'}^{k'}$, $b_i^k < b_{i'}^{k'}$ indicates that subtask k is scheduled earlier than k' on edge server (or local device) a_i^k. Obviously that even for the same $\boldsymbol{A}_i^k$, different $\boldsymbol{B}_i^k$ would also lead to different $avail\{i,k,m\}$, causing different scheduling results. However, it will lock a unique scheduling result when both $\boldsymbol{A}_i^k$ and $\boldsymbol{B}_i^k$ is determined.

Based on the above analysis, we propose our DAG-task offloading problem as follows:

$$\textbf{OPT}: min_{\{\mathbf{A}_i^k,\mathbf{B}_i^k\}}(\sum_{i=1}^{|\mathcal{N}|} EFT_A(i,|V_i|)) \; i\in\mathcal{N} \; k \in V_i \tag{9}$$

$$\textbf{s.t.} Cost \le C, \tag{10}$$

$$a_i^1 = -i, \quad \forall i \in \mathcal{N} \tag{11}$$

$$a_i^{|V_i|} = -i, \quad \forall i \in \mathcal{N} \tag{12}$$

$|V_i|$ denotes the last subtask on device i. Eq.(10) ensures the overall energy consumption of mobile devices is below a threshold, Eq.(11) and Eq.(12) limit that the initial and the last subtasks must be executed locally. The reason is that the two tasks are often responsible for some preparatory and cleaning works on local devices, e.g., getting the users' inputs and echoing the results back to the screen.

B. DAG-task offloading algorithm

The offloading problem is extremely complicated due to the complexity of DAG-task scheduling. To address this problem, we propose a heuristic algorithm for priority-based DAG-task offloading. We first sort the subtasks to ensure that the dependency relationships among the subtasks are guaranteed as long as the subtasks are scheduled with the sorted order. Then, we introduce the priorities for subtasks to perform energy-efficient offloading.

Given a DAG-task, we first rank subtasks as follows:

$$Rank(i,k) = T_{i,k} + \max_{k'\in succ(k)}(Rank(i,k') + C_{kk'}) \tag{13}$$

$succ(k)$ is the set of immediate successor subtasks of subtask k. $C_{kk'}$ is the average data exchange time between subtask k and its subsequent task k', $T_{i,k}$ is the average execution time of subtask k. $C_{kk'}$ and $T_{i,k}$ are calculated as:

$$C_{kk'} = \frac{2\binom{|\mathcal{M}|+1}{2} * data_{kk'}}{2\sum_{m=1}^{|\mathcal{M}|}B_i^m + \sum_{m=1}^{|\mathcal{M}|}\sum_{m'\neq m}B_{mm'}} \tag{14}$$

$$T_{i,k} = \frac{1}{|\mathcal{M}|+1}(T_{i,k}^{-i} + \sum_{m=1}^{|\mathcal{M}|}T_{i,k}^m) \tag{15}$$

It's easy to see that the $Rank(i,k)$ must be larger than $Rank(i,k')$ if k' depends on the output of k. We can guarantee dependency relationships among subtasks as long as we schedule subtasks with higher rank value preferentially. Meanwhile, independent subtasks can be scheduled for execution in parallel.

For task offloading, an intuitive thinking would be that the subtasks with higher computation overhead and local energy comsumption should be offloaded with high priority, thus shorter latency can be obtained and more local energy can be saved. Based on this observation, we further define the priorities of subtasks as follows:

$$Prio(i,k) = \alpha_i Rank(i,k) + \beta_i(\chi_i T_{i,k}^{-i} - \zeta_i\frac{|\mathcal{M}|Size_{i,k}}{\sum_{m=1}^{|\mathcal{M}|}B_i^m}) \tag{16}$$

Algorithm 1 DAG-task offloading algorithm

```
1.Input: $\mathcal{N}$, $\mathcal{M}$, $G_i = (V_i, E_i)$, $data_{kk'}$, $T_{i,k}^m$, $T_{i,k}^{-i}$, $\chi_i$, $\zeta_i$, $\alpha_i$, $\beta_i$, $Size_{i,k}$, $B_i^m$, $B_{mm'}$ $\forall i \in \mathcal{N}$, $\forall k,k' \in V_i$, $\forall m,m' \in \mathcal{M}$
2.Output: The offloading strategies for all the applications on corresponding mobile devices
3.Calculate $Rank(i,k)$ for $\forall\ i \in \mathcal{V}$, $k \in V_i$ by Eq.(13)
4. while there are unscheduled subtasks
5. Select the unscheduled subtask with the highest $Rank(i,k)$ for each $G_i$ to form a candidate scheduling set $\mathcal{A}$
6. Pick subtask $k_{cand}$ with highest $Prio(i,k)$ from $\mathcal{A}$ by Eq.(16)
7. Assign $k_{cand}$ to edge server (or local device) $m$ where minimizes the $EFT_A(i,k_{cand})$ (Eq.(1))
8. Mark $k_{cand}$ as scheduled subtask
9.end while
```

As shown in Eq.(16), we introduce the energy consumption into the priority definition. α_i and β_i are weight coefficients of device i for computation overhead and energy consumption, respectively. Different devices can set these two parameters according to their own conditions, as long as $\alpha_i + \beta_i = 1$ is

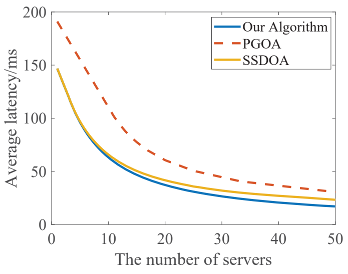

Fig. 2. Average Latency under varying server scales

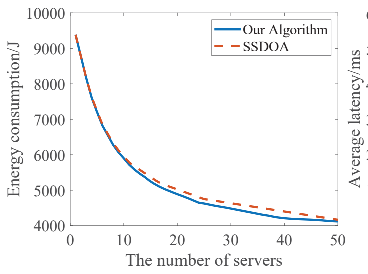

Fig. 3. Energy consumption under varying server scales

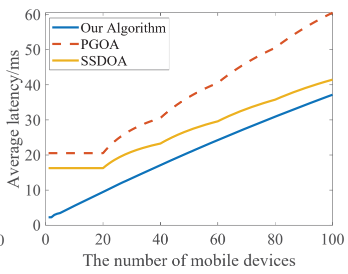

Fig. 4. Average Latency under varying device scales

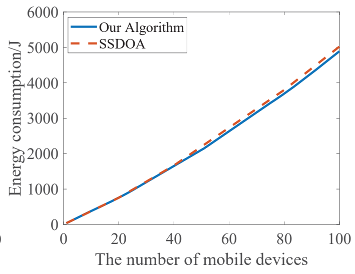

Fig. 5. Average Latency under varying device scales

guaranteed. Note that we ignore the energy consumption of data exchange between subtasks, because the amount of data exchange is often much smaller than the size of task code.

We present our DAG-task offloading algorithm in **Algorithm 1**. We first rank all the subtasks using Eq.(13). Then, for each application on the corresponding devices, we select the unscheduled subtask with the highest rank value to form a candidate scheduling set. The subtask that has the largest rank value among all unscheduled subtasks on one device is selected into the candidate set. We then search the subtask with the highest priority in the candidate set using Eq.(16) and schedule it to corresponding edge server (or local device). The algorithm repeats the above process until all subtasks are scheduled.

Algorithm 1 jointly considers the task latency and energy consumption, offloads the subtasks with large computational overhead and high energy consumption preferentially so as to reduce the total execution delay of applications and the energy consumption of mobile devices as much as possible. Meanwhile, this algorithm is lightweight enough because it avoids the complexity of DAGs by pre-ordering the subtasks. By applying the proposed algorithm in the example shown in Fig. 1, the execution delay is minimized while the dependency among all subtasks is guaranteed.

V. Evaluation

In this section, we evaluate the performance of the offloading algorithm proposed in Section IV. In our simulation scenario, multiple mobile end devices are distributed randomly in an area covered by ultra-dense edge networks, where the edge servers are connected to each other through high-speed fiber networks. The edge servers have heterogeneous computation and communication capabilities. Each device has one DAG-task which is generated randomly. The computational and communicational overhead of each DAG task is randomly assigned. The number of servers and the number of end devices both have significant impact on the end-to-end performance of task offloading. Therefore, we mainly evaluate the impact of the numbers of servers and end devices.

A. The performance of the offloading algorithm under a varying number of servers

We first observe the impact of the number of servers. The evaluation result is shown in Fig. 2 and Fig. 3. We fix the number of devices to 100 and increase the number of edge servers from 1 to 50 to observe the latency and energy consumption of mobile devices. As shown in Fig. 2, the average latency of mobile devices decreases as the number of edge servers grows. It is because more servers provide mobile users with more offloading options, thus subtasks can have a higher degree of parallelism. We also compare our algorithm with other two offloading schemes, the PGOA proposed in [14] and the single-server DAG-task offloading algorithm (SSDOA). The PGOA treats the mobile users' tasks as indivisible tasks, so it cannot obtain the benefits brought by the parallelism of DAG tasks. The SSDOA is a modification of our offloading algorithm, which limits that all subtasks on one device must be offloaded to the same edge server (if they choose to be offloaded), thus it cannot benefit from the edge-server networks. We can also find that the latency gap between SSDOA and our offloading scheme gradually increases as the number of servers grows, which implies that our offloading algorithm can mine the parallelism of computation more effectively. Fig. 3 shows the energy consumption of the algorithms. Note that we do not focus on comparing the energy consumption of PGOA, because this algorithm offloads an entire task to the server for execution, which avoids local computation consumption but comes at the expense of augmented latency. Combining Fig. 2 and Fig. 3, we can see that our algorithm consumes less energy while obtaining a shorter latency.

B. The performance of the offloading algorithm under a varying number of mobile devices

Next, we investigate the impact of the number of mobile devices. The result is shown in Fig. 4 and Fig. 5. We fix the number of edge servers to 20 and increase the number of mobile devices from 1 to 100. We set the DAG-task of every device to be the same to isolate the impact of the number of devices. As shown in Fig. 4, our offloading algorithm achieves the best latency performance among the three algorithms. It is worth noting that the average latency of the PGOA and SSDOA do not change when the number of devices is less than 20. That is because there is no resource contention among devices since there are more servers than devices. However, since a device can only offload to one server, these two algorithms cannot take full advantages of multiple servers. In contrast, our algorithm has no such limitation, thus it better exploits the edge resources. Note that the latency gap between PGOA and SSDOA increases as the number of devices grows. This result explains the benefits of task partitioning. There are more benefits as the number of devices increases because more devices introduce higher parallelism. Fig. 5 shows the energy consumption of these algorithms. Similar to the previous conclusion, our algorithm consumes less energy while providing better latency performance.

VI. Conclusion

In this paper, we investigate the task offloading in ultra-dense edge networks. We describe the general task dependency of applications on mobile devices as directed acyclic graphs (DAGs), then formulate the DAG-task offloading problem which jointly optimizes task latency and energy consumption. In view of the complexity of this problem, we then propose a heuristic priority-based DAG-task offloading algorithm. The algorithm performs task offloading efficiently while guaranteeing task dependency relationships. The evaluation results show that our algorithm can reduce latency and energy consumption in ultra-dense edge networks.

In this work, our performance improvement mainly comes from the maximization of the parallelism between end devices and edge servers. In the future work, we will focus on evaluating the degree of parallelism among subtasks, verifying the performance of our algorithm under different parallelism degrees, and explicitly incorporating parallelism into our offloading strategy.

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), 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 co-corresponding authors.

References

[1] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang and W. Zhao, ”A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications,” in IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1125-1142, Oct. 2017.

[2] Z. Zhao, W. Dong, G. Chen, G. Min, T. Gu and J. Bu, ”Embracing Corruption Burstiness: Fast Error Recovery for ZigBee under Wi-Fi Interference,” in IEEE Transactions on Mobile Computing, vol. 16, no. 9, pp. 2518-2530, 1 Sept. 2017.

[3] W. Gao, W. Du, Z. Zhao, G. Min and M. Singhal, ”Towards Energy-Fairness in LoRa Networks,” in IEEE ICDCS 2019, Dallas, USA, July 7-10, 2019.

[4] D. Liu, Z. Cao, M. Liu, M. Hou, and H. Jinag. ”Contention-Detectable Mechanism for Receiver-Initiated MAC.” ACM Transactions on Embedded Computing Systems (TECS) 18, no. 4 (2019): 31.

[5] P. Mach and Z. Becvar, ”Mobile Edge Computing: A Survey on Architecture and Computation Offloading,” in IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1628-1656, 2017.

[6] 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,” in IEEE Internet of Things Journal, vol. 5, no. 5, pp. 3606-3614, Oct. 2018.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, ”Edge Computing: Vision and Challenges,” in IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637-646, Oct. 2016.

[8] M. Kamel, W. Hamouda and A. Youssef, ”Ultra-Dense Networks: A Survey,” in IEEE Communications Surveys & Tutorials, vol. 18, no. 4, pp. 2522-2545, Fourthquarter 2016.

[9] Y. Zhang, Z. Zhao, C. Shu, G. Min and Z. Wang, ”Embedding Virtual Network Functions with Backup for Reliable Large-Scale Edge Computing,” in Proceedings of IEEE GlobeCom 2018, Shanghai, 2018, pp. 190-195.

[10] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta and D. Sabella, ”On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration,” in IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1657-1681, thirdquarter 2017.

[11] D. Liu, Z. Cao, Y. He, X. Ji, M. Hou, and H. Jiang. ”Exploiting Concurrency for Opportunistic Forwarding in Duty-Cycled IoT Networks.” ACM Transactions on Sensor Networks (TOSN) 15, no. 3 (2019): 31.

[12] C. Shu, Z. Zhao, G. Min and S. Chen, ”Mobile Edge Aided Data Dissemination for Wireless Healthcare Systems,” in IEEE Transactions on Computational Social Systems, Accepted to appear, 2019.

[13] Y. Kao, B. Krishnamachari, M. Ra and F. Bai, ”Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing,” in IEEE Transactions on Mobile Computing, vol. 16, no. 11, pp. 3056-3069, 1 Nov. 2017.

[14] L. Yang, H. Zhang, X. Li, H. Ji and V. C. M. Leung, ”A Distributed Computation Offloading Strategy in Small-Cell Networks Integrated With Mobile Edge Computing,” in IEEE/ACM Transactions on Networking, vol. 26, no. 6, pp. 2762-2773, Dec. 2018.

[15] M. Chen and Y. Hao, ”Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network,” in IEEE Journal on Selected Areas in Communications, vol. 36, no. 3, pp. 587-597, March 2018.

[16] J. Xu, L. Chen and P. Zhou, ”Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks,” in Proceedings of IEEE Conference on Computer Communications, Honolulu, HI, 2018, pp. 207-215.

[17] Z. Ning, P. Dong, X. Kong and F. Xia, ”A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things,” in IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4804-4814, June 2019.

[18] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, ”A Survey on Mobile Edge Computing: The Communication Perspective,” in IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017.

[19] Y. Mao, J. Zhang and K. B. Letaief, ”Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices,” in IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 3590-3605, Dec. 2016.

[20] X. Chen, L. Jiao, W. Li and X. Fu, ”Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing,” in IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 2795-2808, October 2016.

[21] H. Guo and J. Liu, ”Collaborative Computation Offloading for Multiaccess Edge Computing Over Fiber–Wireless Networks,” in IEEE Transactions on Vehicular Technology, vol. 67, no. 5, pp. 4514-4526, May 2018.

[22] K. Zhang et al., ”Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks,” in IEEE Access, vol. 4, pp. 5896-5907, 2016.

[23] Z. Wang, Z. Zhao, G. Min, X. Huang, Q. Ni and R. Wang, ”User mobility aware task assignment for Mobile Edge Computing,” in Future Generation Computer Systems, Volume 85, 2018, Pages 1-8.

[24] X. Lyu and H. Tian, ”Adaptive Receding Horizon Offloading Strategy Under Dynamic Environment,” in IEEE Communications Letters, vol. 20, no. 5, pp. 878-881, May 2016.

[25] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, ”Clonecloud: Elastic execution between mobile device and cloud,” in Proceedings of the 6th Conf. ACM Comput. Syst., 2011, pp. 301–314.

[26] S. E. Mahmoodi, R. N. Uma and K. P. Subbalakshmi, ”Optimal Joint Scheduling and Cloud Offloading for Mobile Applications,” in IEEE Transactions on Cloud Computing, vol. 7, no. 2, pp. 301-313, 1 April-June 2019.

[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, Boston, USA, June 10-13, 2019.