Graph4Edge: A Graph-based Computation Offloading Strategy for Mobile-Edge Workflow Applications

Lingmin Fan[1], Xiao Liu[2], Xuejun Li[*,1], Dong Yuan[3], Jia Xu[1]
[1]School of Computer Science and Technology, Anhui University, Hefei, China
[2]School of Information Technology, Deakin University, Geelong, Australia
[3]School of Electrical and Information Engineering, the University of Sydney, Sydney, Australia
cindyflm@qq.com; xiao.liu@deakin.edu.au; xjli@ahu.edu.cn; dong.yuan@sydney.edu.au; xujia_ahu@qq.com

***Abstract*—Mobile Edge Computing (MEC) expands the capability of mobile devices so that users can run complicated and computation-intensive applications such as workflow and machine learning tasks. Computation offloading is the key technology for MEC and has attracted a lot of research efforts in recent years. However, most of the existing studies employed optimisation algorithms such as GA and PSO which have significant computation overhead. Meanwhile, the computation tasks are often assumed to be independent of each other, which is not applicable to workflow applications with strong task dependencies. To address these issues, we propose Graph4Edge which is a graph-based computation offloading strategy for mobile-edge workflow applications. In this paper, firstly, we formulate the computation offloading problem in MEC using a DAG (Directed Acyclic Graph) based model; secondly, we propose the shortest-path-based algorithm to find the optimal computation offloading plan; finally, preliminary experiments with real-world workflow traces are conducted to evaluate the performance of our proposed strategy. Given the promosing results demonstrated in this paper, we have also presented some important research directions for our future work .**

Keywords—Mobile Edge Computing, Computation Offloading, Workflow, Directed Acyclic Graph, Shortest Path Algorithm

I. Introduction And Related Work

In recent years, with the rapid development of the Internet of Things (IoT), billions of end devices are connectting to the Internet and accessing different services [1]. Cloud Computing can support the requests of a large number of end devices. However, limited bandwidth for accessing Cloud servers over the Internet will result in signficant delays. To effectively solve these problems, a new computing paradigm named Mobile Edge Computing (MEC) has been proposed. Different from the traditional Cloud Computing paradigm, servers located on the network edge in MEC can execute computation tasks close to end devices. Therefore, MEC can reduce the energy consumption of the end device and the task response time through offloading the computation tasks to edge servers.

To effectively utilize the various computing resources in MEC, the task offloading problem has received increasing attention in both academia and industry. The core problem of task offloading is to generate an optimized task offloading decision based on the characteristics of task and computing resource. An task offloading strategy should be able to reduce the energy consumption of the end device and/or task response time effectively. There are many existing studies on task offloading for the MEC environment. According to the main optimization goals, most task offloading strategies can be categorized into either time-focused or energy-focused. On the optimization of time, Xing et al. [2] presented a device-to-device MEC system which can significantly reduce the computation latency of the tasks. On the optimization of the energy consumption of the end device, Zhang et al. [3] proposed an offloading strategy that can minimize the energy consumption for MEC in 5G networks. Chen et al. [4] combined the online task offloading and CPU-cycle frequency scaling for an energy efficiency strategy named TOFFEE which can effectively decrease the energy consumption of the end device. There are also many works which are based on popular optimisation algorithms such as PSO and GA [5, 6]. However, significant computation overhead is a big concern. In addition, most studies only focus on independent tasks which means there is no time or data dependency between tasks. But in fact, task dependency is very common in many real-world applications such as business and scientific workflow applications. Those dependencies are usually represented by some kind of graphs.

There are many work on graph-based algorithms in Cloud Computing and Edge Computing. Yuan et al. [7] proposed a minimum cost storage strategy named CTT-SP which can achieve the best trade-off between computation cost and storage cost for the intermediate datasets in cloud environment. Wang et al. [8] constructed graphs with user application and physical computing resource to optimize cost and proposed an online approximation algorithm to resolve the placement problem of different application components in Edge Computing environments. One of the key constraints needs to be considered for computation offloading is whether the intermediate data need to be transmitted back to the end device. Mach and Becvar [9] assumed that the output data of all tasks need to be transmitted back to the end device. However, it is not necessary in reality. For example, if two consecutive tasks are both offloaded to the same Edge node, the output data of the first task (namely intermediate data) can be used by the second task as input data directly. In such a case, unncessary data transmission time can be saved. Therefore, in this paper, we investigate a more general scenario where not all intermediate data need to be returned. We consider the situation that some tasks may require the input data which is only avaiable at the end device or user input is required at the end device for the next task. In such cases, those tasks cannot be offloaded to the Edge server but only executed locally.

This work is the partially supported by the National Natural Science Foundation of China Project No. 61972001 and the Humanities and Social Sciences of MOE Project No. 16YJCZH048.

To address these issues above, this paper proposes Graph4Edge, a graph-based computation offloading strategy for Mobile-Edge workflow applications. Graph4Edge considers both time and energy and has two prominent advantages: first, it is able to consider the dependencies between tasks which are represented in the graph model; second, it is able to find the optimal offloading decision for the entire workflow by running just once. Details for the proposed graph-based model and algorithms, also the preliminary experimental results and important future works will be presented in the following sections.

II. A Graph-based Computation Offloading Strategy for Mobile-Edge Workflow Applications

A. Concept

Some workflows have many computation-intensive and data-intensive tasks where strong dependencies are existing among them. If two tasks with dependency are executed at different computing nodes, it may result in a long delay for data transmission and a growth for energy consumption of the end device. In contrast, if they are executed at the same computing node, no data transmission is required. Therefore, if the task dependency can be fully utilized, unnecessary data transmission can be avoided to reduce the makespan of workflow and the energy consumption of the end device. In this paper, we firstly build a Workflow Dependency Graph (WDG) based on tasks dependencies.

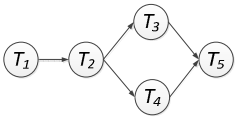

Fig. 1. A simple Workflow Dependency Graph (WDG)

In WDG, each node represents a task. Fig. 1 shows a simple WDG, T_1 points to T_2, which means that there is a dependency between T_1 and T_2. In other words, the output data generated by T_1 serves as the input data of T_2.

WDG is a Directed Acyclic Graph (DAG) with no circles. It describes the dependencies between tasks in the workflow. We denote a task T_i in the WDG as $T_i \in \text{WDG}$, and a set of task $\text{TS} = \{T_1, T_2, \cdots, T_n\}$ in the WDG as $\text{TS} \subseteq \text{WDG}$. To better describe the relationships between tasks in the WDG, we define a symbol $\rightarrow$ which denotes that two tasks have a dependency relationship, where $T_i \rightarrow T_j$ means that T_i is a predecessor task of T_j in the WDG. For example, in the WDG in Fig. 1, we have $T_1 \rightarrow T_2$, $T_2 \rightarrow T_3$, $T_2 \rightarrow T_4$, $T_3 \rightarrow T_5$, $T_4 \rightarrow T_5$. Furthermore, $\rightarrow$ is transitive, where $T_i \rightarrow T_j \rightarrow T_k \Leftrightarrow T_i \rightarrow T_j \wedge T_j \rightarrow T_k \Rightarrow T_i \rightarrow T_k$.

B. Problem Formulation

In this paper, our major goal is to minimize the energy consumption of the end device. If a task is running in a MEC environment, it can be either executed locally or offloaded to the Edge node. Accordingly, the energy consumption model can be divided into the following three types: the transmission energy consumption model, the idle energy consumption model, and the load energy consumption model. As shown in Fig. 2, in order to utilize these energy consumption models, we have defined some important attributes for tasks in WDG.

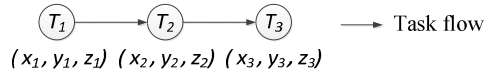

Fig. 2. Three types of energy consumption represented by x, y and z

Specifically, for task T_i, its attributes are defined as $\langle x_i, y_i, z_i, flag_i \rangle$, where the last attribute are omitted in the WDG for simplicity. Here, x_i denotes the transmission energy consumption of the end device when T_i is offloaded to the Edge server. $Comm(T_{i-1}, T_i)$ denotes the data transmission between T_i and T_{i-1} which is the direct predecessors of T_i. $Bandwidth$ is the rate of data transmission. P_{trans} is the data transmission power of end device.

$$x_i = \frac{Comm(T_{i-1}, T_i)}{Bandwidth} \times P_{\text{trans}} \tag{1}$$

y_i denotes the idle energy consumption of the end device when T_i is offloaded to the Edge server, where l_i is the task workload of T_i, f_{edge} is the task processing speed of Edge server, P_{idle} is the idle power of the end device.

$$y_i = \frac{l_i}{f_{\text{edge}}} \times P_{\text{idle}} \tag{2}$$

z_i denotes the load energy consumption of the end device when T_i is locally executed, where f_{end} is the task processing speed and P_{end} is the load power of the end device.

$$z_i = \frac{l_i}{f_{\text{end}}} \times P_{\text{end}} \tag{3}$$

$flag_i$ is a flag which denotes the constraint whether T_i needs to be locally executed or not. If $flag_i = 1$ that means T_i needs to be locally executed because the input data or user confirmation is required at the end device. Otherwise, if $flag_i = 0$ that means T_i can be offloaded to the Edge server.

C. Graph-based Model and Algorithm

In this section, we present the design of Graph4Edge which can find the minimum energy consumption for a given linear WDG with deadline constraint. The core idea is to construct an Energy consumption Transitive Tournament (ETT) based on the WDG. In an ETT, we can guarantee that each path from the start task to the end task has a one-to-one mapping to an feasible offloading strategy. Then we can use the well-known Dijkstra algorithm to find the shortest path in ETT, which will be exactly the offloading strategy with the minimum energy consumption we are seaching for.

The following is the pseudo-code of the algorithm for a linear WDG which means a WDG with no branches namely each task in the WDG only has one direct predecessor and successor. As a work in progress, we will investigate general WDG with branches in the future. For a linear WDG, it has a task set $\text{TS} = \{T_1, T_2, \cdots, T_n\}$ and a local execution task set $localSet = \{T_k | T_k \in \text{WDG} \wedge flag_k = 1\}$.

For task T_i, it has out-edges to all the tasks in the set of $\{T_j | T_j \in \text{WDG} \wedge T_i \rightarrow T_j\}$, and in-edges from all the tasks in the set of $\{T_k | T_k \in \text{WDG} \wedge T_k \rightarrow T_i\}$ (line 3). And for the set of $\{T_i | T_i \in localSet\}$, we prune the edges that T_i serve as the head or tail (line 4). Formally, $T_i, T_j \in \text{WDG} \wedge T_i, T_j \notin localSet \wedge T_i \rightarrow T_j \Rightarrow \exists e \langle T_i, T_j \rangle$. We set weight to each edge in

ETT (line 5). For an edge $e\langle T_i, T_j\rangle$, we denote its weight as $\omega\langle T_i, T_j\rangle$, which is defined as the sum of the idle energy consumption of T_i and T_j and the transmission energy consumption of the successor of T_i and T_j and the load energy consumption of the tasks between T_i and T_j, supposing that only T_i and T_j are both offloaded to the Edge server. Formally,

$$\omega\langle T_i, T_j\rangle = y_i + y_j + x_{i+1} + x_j + \sum_{\{T_k | T_k \in \mathrm{WDG} \wedge T_i \to T_k \to T_j\}} z_k \quad (4)$$

Algorithm: Graph4Edge

Input: a workflow dependency graph (WDG); //Not include T_s and T_e
a set of local-execution tasks in the WDG $localSet$;
the deadline constraint $Time_{ddl}$;

Output: a set of tasks in the WDG S; //The offloading decision

1: Compute $(x_i y_i z_i)$ for all tasks by formula (1) (2) (3);
2: Add the two virtual tasks into WDG, T_s before first task T_1 and T_e after last task T_n, and set $x_s = y_s = z_s = 0$ and $x_e = y_e = z_e = 0$;
3: Add new directed edges for each task in WDG that start with it and point to all subsequent tasks to construct ETT;
4: Prune the excess edge $e\langle T_i, T_j\rangle$ if T_i or T_j in $localSet$;
5: Compute weight for each edge in ETT by formula (4);
6: for ($k = n$+1 to 1) do // $T_e = T_{n+1}$
7: $P_{\min}\langle T_s, T_k\rangle$ = Dijkstra_Algorithm(T_s, T_k, ETT);
8: S = set of tasks that $P_{\min}\langle T_s, T_k\rangle$ traversed;
9: if (workflow makespan < $Time_{ddl}$) then
10: end for
11: else
12: $k = k - 1$;
13: end if
14: end for
15: if ($k = 0$) then
16: S = null;
17: end if
18: return S;

Fig. 3 demonstrates a simple example of constructing an ETT for a WDG that only has three tasks, where T_s is the start task that only has out-edges and T_e is the end task that only has in-edges, and supposing $localSet = \{T_3\}$.

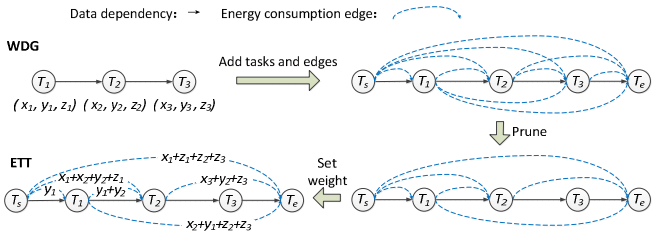

Fig. 3. An example of constructing an ETT

In the pseudo-code lines 6 to 14 of Graph4Edge algorithm, there is a loop to make sure that the offloading strategy can satisfy the workflow deadline constraint. Hence, we can find the shortest path from T_s to T_k of the ETT meeting the deadline. From the construction steps, we can clearly see that ETT is an acyclic complete oriented graph. Hence we can use Dijkstra algorithm to find the shortest path from T_s to T_k. The Dijkstra algorithm is a classic greedy algorithm to find the shortest path in graph. We denote the shortest path from T_s to T_k as $P_{\min}\langle T_s, T_k\rangle$, and put the tasks that $P_{\min}\langle T_s, T_k\rangle$ traverses into the empty set S. The offloading decisions for tasks in S are offloaded to Edge server while the remaining tasks which are not in S will be locally executed. In the worst case (line 15), if $k = 0$, then the offloading decisions of all tasks in the workflow are locally executed, namely no offloading.

III. Evaluation

In this section, some preliminary experiments are conducted to evaluate the performance of our proposed Graph4Edge algorithm. In these experiments, we employ the Graph4Edge algorithm for offloading strategy and the Min-Min algorithm for task scheduling to resolve resource conflicts. Firstly, the influence of user's deadline constraint on Graph4Edge algorithm is analyzed. Secondly, the performance of Graph4Edge on optimizing the energy consumption of the end device is evaluated. Finally, we compare Graph4Edge with three algorithms including Baseline, PSO [5] and GA [6]. The Baseline algorithm is to make the offloading decision by comparing the offloading energy consumption with the local execution energy consumption of the end device for each task.

A. Experimental environment

All experiments are conducted on FogWorkflowSim [10] which is a workflow simulator for Fog/Edge computing providing real-world workflow models and traces, and running on a server with the following configurations: Intel core i7, dual-core 3.2GHz CPU, 16 GB RAM.

The parameter settings for the MEC environment are described in TABLE I. The MEC environment has two layers: the end device layer and the Edge server layer. In the experiments, we configure the MEC environment with 3 Edge servers and 1 end device. The rate of data transmission between end devices and Edge servers is set as 100Mbps. As in this paper we only consider the energy consumption of the end device, the power for Edge servers are set as N/A.

TABLE I. Parameter settings of MEC environment

Parameters	End Device	Edge Server
MIPS	1000	1300
Load Power (mW)	700	N/A
Idle Power (mW)	30	N/A
Data Transmission Power (mW)	100	N/A

For PSO algorithm, the particle number is 30 and inertia weight is 1. The learning factors C1 and C2 are both 2. For GA algorithm, the population size is 50. The rates of crossover and mutation are 0.8 and 0.1 respectively. The iteration numbers of PSO and GA are both 100.

Only linear workflow is used in the experiment, that is, the tasks only have sequence dependency. We generate the workflow size randomly with tasks between 10 to 50. For each task, we consider the number of CPU cycles is between 1000 and 15000 Megacycles randomly, and the size of input and output data is between 0 and 1500 MB randomly. The $localSet$ is also generated randomly and its size is set as 20% of the total number of tasks, namely 20% of the tasks need to be locally executed.

B. Results

The deadline constraint of the workflow is set between 70% and 130% of the total task completion time executed at the end device. Fig. 4 shows results on energy consumption and the offloading task number under different deadlines. It shows the

average value of ten experiments each has the same workflow but under different deadline constraint. It can be seen that when the deadline constraint increases, the offloading task number also increases but the energy consumption is decreasing. This indicates that with the increase of the deadline constraint, the offloading decisions generated by Graph4Edge algorithm has increasing effect on reducing energy consumption. However, when the deadline constraint reaches to 120% and over, the effect becomes flat. Therefore, in the following experiments, we focus on the situation when the deadline is 120%.

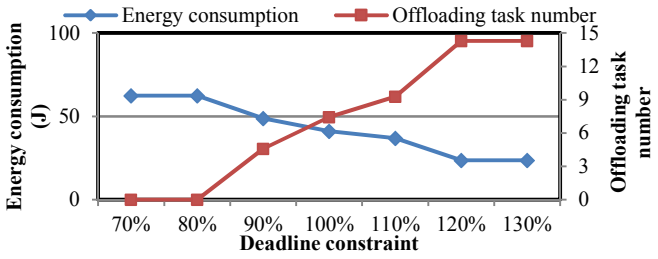

Fig. 4. Energy consumption and offloading task number with different deadline constraint

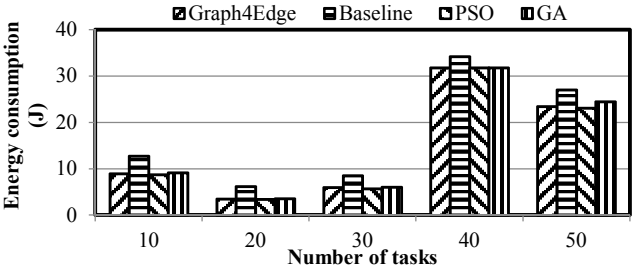

Fig. 5. Comparison of the energy consumption of end device

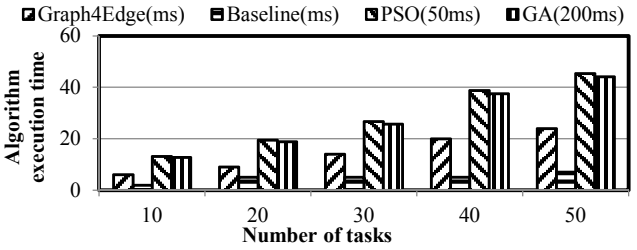

Fig. 6. Comparison of the algorithm execution time

Fig. 5 and Fig. 6 illustrates the results on energy consumption and algorithm execution time. It can be seen in Fig. 5 that Graph4Edge is much better than Baseline in energy consumption. In Fig. 6, noted that the time unit for Graph4Edge and Baseline is millisecond (ms), while PSO is 50ms and GA is 200ms. Therefore, PSO and GA are much slower compared with Graph4Edge. Clearly, Graph4Edge is comparable to PSO and slightly better than GA in energy consumption, but it is running 100 times faster than PSO and 400 times faster than GA. More importantly, Graph4Edge can find the optimal solution by design.

IV. Conclusion and Future Work

In this paper, we proposed the Graph4Edge model and algorithm for computation offloading of Mobile-Edge workflow applications. Based on the WDG and ETT model, we proposed an alogirhtm that can find the best offloading strategy for linear workflows. Experiments conducted in the FogWorkflowSim platform has shown very promising results compared with other representative algorithms.

Based on the promising results that we obtained so far, we will further enhance our Graph4Edge model in the following directions. First, we will investigate the offloading tasks that are represented by the general DAG graph, since many applications in the real world have more complicated task dependencies, e.g., many tasks can be processed in parallel. Second, we will extend our Edge-based tasks offloading model to incorporate cloud computing resources. Cloud resources are needed for tasks offloading due to various reasons, e.g., the limited capacity of Edge servers to handle temporal heavy workload, more energy efficiency for using the cloud, users moving out of the coverage of Edge servers, etc. We will design a three-tier tasks offloading model, i.e., device, Edge and cloud, which is more generic for the real-world applications. Based on the enhanced Graph4Edge model, we must design a new ETT graph to represent the energy consumption, where solving the optimal solution will become NP-Hard. The new problem will be in different forms according to the constraints from the real world, which makes this research a great potential to be applied in practice.

References

[1] T. Zhang, J. Jin, X. Zheng and Y. Yang, "Rate Adaptive Fog Service Platform for Heterogeneous IoT Applications," *IEEE Internet of Things Journal*. 2019.

[2] H. Xing, L. Liu, J. Xu and A. Nallanathan, "Joint Task Assignment and Resource Allocation for D2D-Enabled Mobile-Edge Computing," *IEEE Transactions on Communications*. vol. 67, pp. 4193 - 4207, 2019.

[3] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan and Y. Zhang, "Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks," *IEEE Access*. vol. 4, pp. 5896-5907, 2016.

[4] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu and X. S. Shen, "TOFFEE: Task Offloading and Frequency Scaling for Energy Efficiency of Mobile Devices in Mobile Edge Computing," *IEEE Transactions on Cloud Computing*. 2019.

[5] A. Kaswan, V. Singh and P. K. Jana, "A Multi-Objective and PSO Based Energy Efficient Path Design for Mobile Sink in Wireless Sensor Networks," *Pervasive and Mobile Computing*. vol. 46, pp. 122-136, 2018.

[6] H. Hallawi, J. Mehnen and H. He, "Multi-Capacity Combinatorial Ordering GA in Application to Cloud Resources Allocation and Efficient Virtual Machines Consolidation," *Future Generation Computer Systems*. vol. 69, pp. 1-10, 2017.

[7] D. Yuan, L. Cui, W. Li, X. Liu and Y. Yang, "An Algorithm for Finding the Minimum Cost of Storing and Regenerating Datasets in Multiple Clouds," *IEEE Transactions on Cloud Computing*. vol. 6, pp. 519-531, 2018.

[8] S. Wang, M. Zafer and K. K. Leung, "Online Placement of Multi-Component Applications in Edge Computing Environments," *IEEE Access*. vol. 5, pp. 2514-2533, 2017.

[9] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys & Tutorials*. vol. 19, pp. 1628-1656, 2017.

[10] X. Liu, L. Fan, J. Xu, X. Li, L. Gong, J. Grundy and Y. Yang. "FogWorkflowSim: An Automated Simulation Toolkit for Workflow Performance Evaluation in Fog Computing."[Online]. Available: https://2019.ase-conferences.org/track/ase-2019-Demonstrations.