2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

Computation Offloading for Mobile-Edge Computing with Multi-user

Luobing Dong, Meghana N. Satpute, Junyuan Shan, Baoqi Liu, Yang Yu, Tihua Yan,

***Abstract*—New age smartphones are equipped with high processing power and internet connectivity. Hence, smartphones are capable of executing applications, which were only possible by desktops or laptops until recently. Some examples of such applications are email, banking, flight booking etc. People prefer to use mobile devices for these applications due to the usability and portability of mobile devices. However, because of hardware limitations, mobiles have limited resources such as battery life, power and capacity. Researchers are constantly looking for ways to maximize the usage of these resources. The execution of any application on mobile, needs storage capacity of mobile to store, battery life of mobile to keep running and processing capacity of mobile to process. Thus, more resources are needed to run more applications on these devices. To reduce the load of applications on mobile devices and use the resources efficiently, it is necessary to move some load of applications to remote cloud server in such a way that the applications will run seamlessly. Computation offloading for mobile-edge computing (MEC) is a mechanism to utilize mobile resources well by moving resource-intensive applications to cloud server at network edge. In the case of multiple users, the total computing capacity of the server needs to be taken into consideration for allocating resources to multiple users. The key to efficient computation offloading is allocating applications to mobile and remote server in such a way that minimizes transmission energy. In this paper, we formulate the computation offloading problem as graph cut problem and propose a solution based on spectral clustering computation. First, for the applications on mobile a corresponding network flow graph model is defined. Then, label propagation theory is applied on the network graph and the network graph is simplified by compressing and combining. Finally, the optimal solution is obtained by computation using spectral clustering algorithm. Experiments show that the algorithm is effective in handling programs with loosely coupled as well as highly coupled functions.**

Keywords — Mobile-edge computing, Computation Offloading, Multiple users, Spectral Graph Theory.

I. INTRODUCTION

Smart devices have become integral part of the daily lives of people. Applications such as emails, videos, games, shopping, social networking can be easily accessible with the touch of a fingertip on mobile screen. Smart phones are embedded with new technologies such as face recognition, natural language processing, interactive games, virtual reality etc. These technologies provide rich user experience. Due to mobile devices' size and weight, mobile terminals are limited in processing power, storage capacity, network connectivity, computing resources and battery capacity. In addition, demanding applications consume high battery power. This is a serious obstacle that limits users from fully utilizing their devices. A recent survey showed that nearly half of responders were dissatisfied with the battery power of their mobile phones. They need twice as much power as they have now [8]. The trade-off between resource-intensive applications and resource-constrained mobile devices pose challenge for mobile platform development [2]. These factors motivate the concept of transferring or offloading some applications from mobile to remote cloud.

In last decade, Mobile Cloud Computing (**MCC**) became popular because it helped to reduce load on mobiles by transferring computationally exhausting applications to remote cloud server [3]. But transferring applications to remote servers result in adding latency and security or privacy issues. This led to think researchers about other possible options such as **MEC** [6, 17], cooperate computing [24], etc. One milestone in this direction is transferring some applications from mobile and run those on server at the network edge with cloud server like capabilities and services [6]. This technique is called **MEC**. **MEC** brings several advantages over **MCC** such as: 1) achieving lower latency, 2) having better privacy and security for mobile applications, 3) saving more energy for mobile devices, and 4) supporting context-aware computing [13]. Therefore, **MEC** is a promising technology for expanding mobile terminal resources and capability.

Until now, much progress has been made in the research of computation offloading methods. Partitioning applications and offloading large scale computing task to the edge is a promising direction in this research [19]. Ou and others proposed an adaptive partitioning algorithm for partitioning application into offloadable and unoffloadable partitions.. Experiments indicated the efficiency and cost-effectiveness of this technique [14]. Liu and others proposed a Dynamic Programming based Offloading Algorithm (**DPOA**) [11] that can quickly find the optimal partition among the executable sub-components of mobile applications. A crucial point here is that, calculating offloadable partition of application

Luobing Dong is with the School of Computer Science and Technology, Xidian University, Xi'an, China. Luobing Dong is corresponding author (e-mail: lbdong@xidian.edu.cn).

Meghana N. Satpute is with the department of Computer Science, The University of Texas, Dallas, USA. (e-mail: mns086000@utdallas.edu)

Junyuan Shan is with the School of Computer Science and Technology, Xidian University, Xi'an, China (e-mail: 1213542673@qq.com).

Baoqi Liu is with the School of Computer Science and Technology, Xidian University, Xi'an, China (e-mail: 1908952076@qq.com).

Yang Yu is with the School of Computer Science and Technology, Xidian University, Xi'an, China (e-mail: 2500256383@qq.com).

Tihua Yan is with the School of Computer Science and Technology, Xidian University, Xi'an, China (e-mail: thyan@mail.xidian.edu.cn).

2575-8411/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDCS.2019.00088

requires access to resource-rich servers over wired or wireless networks for a short time [1]. These servers can use virtualization to provide the ability to compute partitions in isolation and protect data. Based on the analysis of the average delay of data transmission and the average power consumption of mobile devices, [10] proposed an efficient one-dimensional search algorithm, which can find out the optimal task scheduling strategy. Methods [16], [23], and [22] are some other common methods for selecting tasks to offload.

However, all these known methods continue to have their own limitations. First, most of them partition the original program into coarse modules like components [10], layers [9], etc. This can effectively reduce the amount of calculation of the partition algorithms. But partitioning into course modules can lead to inaccuracy in some cases. For example, there may be a component that cannot be offloaded to the edge server just because it has a function that needs to collect data directly from local sensors. It may also have some other functions with lot of calculation and rare communication with other functions. The offloading of such functions might be beneficial to the system from computation point of view. However, the function cannot be offloaded and hence the whole component cannot be offloaded. Second, many proposed heuristics and approximation methods can get the partition fast, but there are some inaccuracies involved.

In this paper, we consider the computation offloading problem for **MEC** system with multi-user (**COPMECS**) based on function level. The original application is represented as function call graph. Various factors are taken into consideration before offloading. The optimal offloading decision depends on the balance between computation and communication. The final decision is made by considering all computing nodes at the same time. The graph is first compressed in size by merging highly coupled nodes. Then we modelled the offloading problem as the minimum cut searching problem of the compressed graph. Finally, the spectral clustering method is used to get the accurate offloading scheme.

The following are the detailed contributions of this paper:

- We model the original application as its function call graph and formulate the **COPMECS** as a constrained double-objective optimization problem.
- We design a novel function call graph compression method. Considering the amount of computation of each function and the amount of communication between each pair of functions, we design a special label rule. Based on this rule, we divide the function call graph into multiple sub-graphs using label propagation theory. The nodes in each sub-graph that satisfy the compression condition are merged to compress the original graph.
- After transferring the objective function to the minimum cut searching problem of the compressed graph, the spectral clustering method is introduced to get the accurate offloading scheme.
- Experiments are conducted using Spark framework to prove the accuracy and efficiency of our novel algorithms which is better than heuristic algorithms.

Rest of the paper is organized in following manner. In Section 2, system model definition and objective formulation is discussed. In Section 3, we introduce our label rule and label propagation theory-based function call graph compression algorithm. The spectral clustering method is also introduced to get the accurate computation offloading scheme. In Section 4, we provide the experiments that compare the performance of our novel algorithms with some traditional algorithms. Finally, Section 5 concludes the paper.

II. System Model and Problem Formulation

As already mentioned above, we consider the **COPMECS** based on function level. Therefore, we must generate the function data flow graph of the application first. Unfortunately, we usually can only get the compiled executable file of normal applications. Here, we use Soot [20] to get the internal functions and their calling relationships from the compiled executable of the application. Then we can easily generate the function data flow graph. However, not all functions in the application program are suitable for remote execution. Some funcitons participate in large amount of data exchange with other functions and their execution highly depends on local data interaction like sensors' data reading, local I/O devices accessing, etc. We call these functions as unoffloaded functions. These functions will be excluded from the function data flow graph.

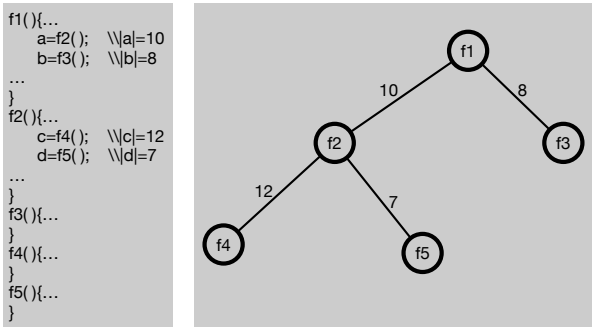

Figure 1. An function data flow graph example of a program

We use weighted un-directed graph $G^i = (V^i, F^i)$ to represent the function data flow graph of a mobile application A^i of user u^i. We assume that all users are served by one single edge server S. $V_i = \{v_1^i, v_2^i, ..., v_n^i\}$ is the node set of G^i with each node v_j^i represents the jth function of A^i. F^i is the edge set of G^i. If function v_j^i needs to exchange data with v_l^i, there will be an edge between

v_j^i and v_l^i. we use $f_{j,l}^i$ to represent the edge between v_j^i and v_l^i. The weight of each node represents the amount of computation of its corresponding function. The weight of each edge represents the amount of communication between the functions corresponding to the two ends of this edge. We use V_c^i to denote the functions of A^i that are needed to be executed locally, and V_s^i to denote the functions of A^i that can be offloaded to the edge server. Obviously, $V^i = V_c^i \cup V_s^i$ and $V_c^i \cap V_s^i = \emptyset$. Figure 1 shows an example of function data flow graph of a program.

We assume that t_c^i represents the computing time for all functions that should be computed locally. t_s^i represents the computing time for all functions that should be computed remotely. I_c^i represents the available computing resources of the mobile device of user u^i. I_s^i represents the available computing resources of u^i assigned by S. Then we can calculate t_c^i and t_s^i by formula (1) and formula (2). Here w_j^i represents the weight of the node v_j^i. Here ${}^w\mathrm{t}_j^i$ represents the time consumed by v_j^i when waiting for the resource allocated by S.

$$t_c^i = \frac{\sum_{v_j^i \in V_c^i} w_j^i}{I_c^i} \tag{1}$$

$$t_s^i = \frac{\sum_{v_j^i \in V_s^i} w_j^i}{I_s^i} + {}^w\mathrm{t}_j^i \tag{2}$$

We use p_c^i to denote the unit power consumption of user u^i. e_c^i represents the energy consumption of local computation of user u^i. We can calculate e_c^i by formula (3).

$$e_c^i = t_c^i \cdot p_c^i \tag{3}$$

We can get the transmission energy consumption e_t^i between V_c^i and V_s^i as formula (4) and the transmission time consumption between them as formula (5).

$$e_t^i = \frac{\sum_{v_j^i \in V_c^i} \sum_{v_l^i \in V_s^i} s(v_j^i, v_l^i) \cdot p_t^i}{b^i} \tag{4}$$

$$t_t^i = \frac{\sum_{v_j^i \in V_c^i} \sum_{v_l^i \in V_s^i} s(v_j^i, v_l^i)}{b^i} \tag{5}$$

Here p_t^i represents the unit energy consumption of transmission from user u^i to S. $s(v_j^i, v_l^i)$ denotes the weight of edge $f_{j,1}^i$. b^i denotes the transmission bandwidth between u^i and S. For the simplicity of discussion, we assume that $\forall u^i$, $b^i = b$, $p_s^i = p_s$, and $p_c^i = p_c$.

Considering all computing requirements from all users at the same time, the offloading scheme should be decided based on the consumption balance between local computation and communication. Therefore, our objective function can be formulated by formula (6). $E = \sum_i e_c^i + \sum_i e_t^i$ represents the total computing energy consumption of the system. $T = \sum_i t_c^i + \sum_i t_s^i + \sum_i t_w^i$ represents the total transmission energy consumption of the system.

$$\begin{cases} min(E) \\ min(T) \end{cases} \tag{6}$$

III. Offloading Scheme based on Graph Compression and Graph Spectrum

Generally, the unit energy consumption for wireless transmission (p_t^i) is far larger than the unit energy consumption of local computing (p_c^i). Therefore, The value of E mainly depends on $\sum_i e_t^i$. If we want to minimize E, we should reduce communication which results from offloading. At the same time, the value of T mainly depends on $\sum_i t_c^i$ and $\sum_i t_s^i$. We know that, the resources of edge servers are always limited because of the construction cost. Too much offloading will inevitably increase the load of S, and then $\sum_i t_s^i$ or transmission energy consumption will also increase significantly. But too small offloading will increase $\sum_i t_c^i$ or local energy consumption. Therefore, we should balance between local computation and communication to achieve the optimal offloading scheme. On the other hand, we consider the problem based on the function level. This will make the size of the data flow graph exponential.

In this section, we design a novel offloading scheme. First, we compress the size of the function data flow graph. Second, we transfer the objective problem to the minimum cut searching problem and introduce the spectral graph theory to solve it. Finally, we get the optimal offloading scheme based on greedy algorithm.

A. Graph Compression

In this section, we introduce the label propagation algorithm [26] (**LPA**) to compress the size of the original function data flow graph effectively. Considering the transmission consumption between functions, we design a special label rule which can lead the label propagation process to find highly coupled functions. Then we split the graph into component based sub-graphs and design a parallel label propagation process. Finally, we define nodes merging rule to compress highly coupled functions based on the clustering results of the parallel label propagation process.

LPA is a semi-supervised learning method for graph clustering which was proposed in 2002. It randomly assigns labels to some nodes in the graph and propagate these labels to the unlabeled nodes throughout the course of the algorithm. Nodes with the same label will be seen as belonging the same class at the end. It has received widespread attention from scholars because it is easy to implement [15]. In the following, we will present our compression algorithm in details.

Graph Partition: The objective of compression is to reduce the size of the function data flow graph by merging some highly coupled functions. However, the coupling degree of two functions from two different components

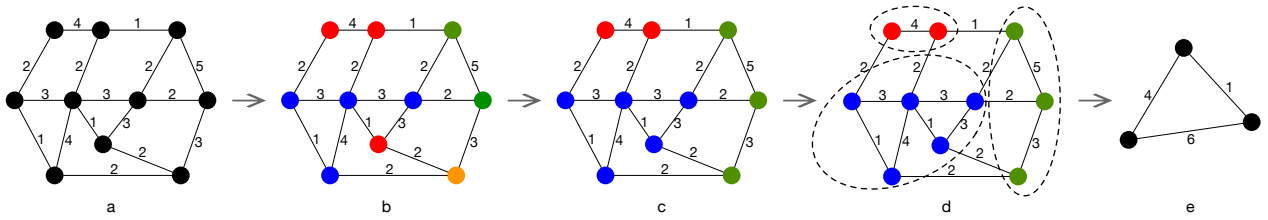

Figure 2. An example of sub-graph compression. a, original graph. b, first round of propagation. c, second round of propagation. d, clustering result. e, compression result.

must be small. Therefore, the functional data flow sub-graph corresponding to each component can perform compression independently. In the beginning, we divide the original functional data flow graph into a set of sub-graphs based on component boundaries. We use ${}^{c}\mathrm{G}^{i} = (G_1^i, G_2^i, \cdots, G_m^i)$ to represent the partition result of A^i with each G_j^i represents a sub-graph corresponding to one component.The following steps will be performed in parallel.

Label initialization and propagation: It is difficult to guarantee the availability of the final classification result by randomly assigning the initial label. In order to accelerate the process of propagation, we choose the node which has the maximum out-degree as the starter node of the propagation. After label propagation, we aim to have same label for functions with highly coupling degree. Here we use the weight of an edge to denote the coupling degree between the two functions associated with this edge. We set a weight threshold w. If the weight of an edge associated with a labeled node is larger than w, and the another end of this edge is unlabeled, the unlabeled node will be given the same label, Otherwise, it will be given different label. Starting with the starter, all nodes will be labeled one by one according to depth-first or breadth-first policies. We call this process a propagation process. As the propagation continues, more and more functions will be labeled. This process will be repeated until the end condition is met.

End of propagation: We set two end conditions for the propagation process. If either one is satisfied then the whole process will end. First, we define the rate of label updating α as formula (7). $update_{num}$ represents the number of functions whose label were updated in one propagation process. $total_{num}$ represents the total function number of sub-graph G_j^i. When $\alpha \leq \alpha_t$, the whole process will be terminated. α_t is the pre-set threshold of α. Second, we set the highest iteration number β_t, when the total number of propagation processes is larger than β_t, the whole process will be terminated.

$$\alpha = \frac{update_{num}}{total_{num}} \tag{7}$$

Compression: After the label propagation, the original graph G^i will be partitioned into many different clusters. Nodes with the same label will be in the same cluster. Nodes with different labels will be in the different clusters. Any two nodes which are in the same cluster and are connected directly will be merged into one node. This merging process is the so-called compression. This compression process can guarantee that highly coupled functions be executed in the same device (U^i or S). This can avoid the offloading which can generate huge transmission cost.

Figure 2 shows an example of a sub-graph compression. From figure 2a to figure 2c the original sub-graph experiences two propagation processes. figure 2d shows the nodes which can be merged and figure 2e shows the result after compression. The size of the sub-graph reduced from 10 to 3.

All details of the graph compression algorithm are shown in Algorithm 1. All un-offloadable functions are removed from the original graph in Line 1. The graph is split into some smaller graphs based on the boundary of components (Line 2 to Line 4). For each sub-graph, the node with the largest out-degree is found and labeled as the starter first (Line 6). One new process will be generated for each sub-graph (Line 6). All propagation processes will be executed in parallel. The label propagation process starts from the starter until one end condition is satisfied (Line 8 to 15). The functions with the same label and connecting directly to each other will be compressed in each labeled sub-graph (Line 16).

B. Graph Spectrum based Offloading Scheme Generation

In this section, we design an offloading scheme generation algorithm. First we transfer the original object to a series of graph minimum cut searching problem. Second, we introduce the Spectral Graph Theory to get the minimum cut of sub-graphs. Finally, we design an offloading scheme generation algorithm based on greedy strategy.

Object transmission: As already mentioned, we aim to minimize the energy consumption of the system which mainly depends on the transmission that results from offloading. As we all know, the communication between two devices includes the data transmission and some necessary control messages transmission. The amount of control messages transmission depends on the number of data transmission other than the amount of data that needs to be transferred. Sometimes, a series of smaller data transmission may

consume more energy than one bigger data transmission. Therefore, we should control the number of transmission that results from offloading.

We assume that ${}^c_r\mathrm{G}^i_j$ represents the compressed graph of sub-graph ${}^c\mathrm{G}^i_j$. After the graph compression process, we get a set of compressed sub-graphs ${}^c_rG^i = ({}^c_r\mathrm{G}^i_1, {}^c_r\mathrm{G}^i_2, \cdots, {}^c_r\mathrm{G}^i_n)$. The weight of any edge of any sub-graph ${}^c_r\mathrm{G}^i_j$ is usually not too large because of the compression. To reduce the number in the communication of the final offloading scheme, we just partition each sub-graph ${}^c_r\mathrm{G}^i_j$ into two parts such that they have the minimum communication amount. One part executes locally, and another part executes remotely. In other words, we need find the minimum cut of each sub-graph ${}^c_r\mathrm{G}^i_j$.

Graph spectrum based minimum cut searching: We assume that one compressed sub-graph ${}^c_r\mathrm{G}^i_j$ is partitioned into two parts ${}^c_r\mathrm{G}^i_{j,1}$ and ${}^c_r\mathrm{G}^i_{j,2}$. The cut corresponding to these two parts $CUT({}^c_r\mathrm{G}^i_{j,1}, {}^c_r\mathrm{G}^i_{j,2})$ represent as formula (8).

$$CUT({}^c_r\mathrm{G}^i_{j,1}, {}^c_r\mathrm{G}^i_{j,2}) = \sum_{v^i_j \in {}^c_r\mathrm{G}^i_{j,1}, v^i_l \in {}^c_r\mathrm{G}^i_{j,2}} s(v^i_j, v^i_l) \quad (8)$$

Our object in this step is to find the minimum cut of ${}^r_c\mathrm{G}^i_j$. It can be formulated as formula (9).

$$\arg\min_{{}^r_c\mathrm{G}^i_{j,1}, {}^r_c\mathrm{G}^i_{j,2}} CUT({}^r_c\mathrm{G}^i_{j,1}, {}^r_c\mathrm{G}^i_{j,2}) \quad (9)$$

Methods based on the graph spectrum theory are popular in a wide range of applications like data analysis, graph clustering, and applied mathematics due to their strong underlying theory and good performance [12]. This theory is based on the eigenvalues and eigenvectors of the Laplace matrix which is associated with the graph [5]. The spectral cut features of the graph can be quantified based on these eigenvectors. We introduce this theory to search the minimum cut of each compressed sub-graph. We assume that q_i=$\begin{cases} 1 & i \in {}^r_c\mathrm{G}^i_{j,1} \\ -1 & i \in {}^r_c\mathrm{G}^i_{j,2} \end{cases}$, L represents the Laplace matrix of ${}^r_c\mathrm{G}^i_j$. We can get Theorem 1.

Theorem 1. *The minimum cut of ${}^r_cG^i_j$ equals to its second smallest eigenvalue.*

To prove Theorem 1, we should know two other theorems first.

Theorem 2. *For any cut of graph ${}^c_rG^i_j$ and some constants d_1 and d_2, formula (10) is true.*

$$CUT({}^r_c\mathrm{G}^i_{j,1}, {}^r_c\mathrm{G}^i_{j,2}) = \frac{q^T L q}{(d_1 - d_2)^2} \quad (10)$$

Theorem 3. *Each extreme point of $CUT({}^c_rG^i_{j,1}, {}^c_rG^i_{j,2})$ is corresponding to one eigenvector of L. Here L is the Laplace matrix of ${}^c_rG^i_j$.*

Algorithm 1 Graph Compression Algorithm

Input $G = (G^1, G^2, .., G^n), n\_max, r\_min$
OutPut: ${}^c_r\mathrm{G}$
$G^i = remove\_unoffloaded(G^i)$
for all G^i **do**
  ${}^c\mathrm{G}_i = componentSplit(G^i)$
end for
for all ${}^c\mathrm{G}^i_j \in {}^c\,\mathrm{G}_i$ **do**
  create new process
  $v = Largest\_outdegree({}^c\mathrm{G}^i_j)$
  $total\_num = 0$
  $update\_num = 0$
  $rate = 1$
  while $total\_num < n\_max$ *or* $rate > r\_min$ **do**
    $update\_num, {}^c\,\mathrm{G}^i_j = label\_propagation(v)$
    $rate = \frac{update\_num}{node\_num({}^c_r\mathrm{G}^i_j)}$
    $total\_num = total\_num + 1$
  end while
  ${}^c_r\mathrm{G}^i_j = compress({}^c\mathrm{G}^i_j)$
  $Inesrt({}^c_r\mathrm{G}^i_j, {}^c_r\,\mathrm{G})$
end for
return ${}^c_r\mathrm{G}$
=0

Algorithm 2 Offloading Scheme Generation

Input $G = (G^1, G^2, .., G^n), n\_max, r\_min$
OutPut: V_1, V_2
${}^c_r\mathrm{G} = Graph\_Compress(G, n\_max, r\_min)$
for all ${}^c_r\mathrm{G}^i_j \in {}^c_r\,\mathrm{G}$ **do**
  $\lambda = second\_smallest\_engivalue({}^c_r\mathrm{G}^i_j)$
  ${}^c_r\mathrm{G}^i_{j,1}, {}^c_r\,\mathrm{G}^i_{j,2} = split({}^c_r\mathrm{G}^i_j, \lambda)$
  $Inesrt({}^c_r\mathrm{G}^i_{j,1}, {}^c_r\,\mathrm{G}^i_{j,2}, V_2)$
end for
$Inesrt(V_2^{'}, V_1)$
$Remove(V_2^{'}, V_2)$
$E_1 = T_1 = \infty$
$E_1 = energy\_comsup(V_1, V_2)$
$T_1 = time\_comsup(V_1, V_2)$
while $E_t + T_t < E_{t-1} + T_{t-1}$ **do**
  for all $V^i_2 \in V_2$ **do**
    $E^i_t = energy\_comsup(V_1 + V^i_2, V_2 - V^i_2)$
    $T^i_t = time\_comsup(V_1 + V^i_2, V_2 - V^i_2)$
  end for
  $v = find\_smallest(E^i_t + T^i_t)$
  $Inesrt(V^i_2, V_1)$
  $Remove(V^i_2, V_2)$
  $t = t + 1$
  $E_t = E^i_t$
  $T_t = T^i_t$
end while
return V_1, V_2 =0

$$\lambda_{min} < min_{{}^r_c\mathrm{G}^i_{j,1}+{}^r_c\mathrm{G}^i_{j,2}={}^r_c\mathrm{G}^i_j} CUT({}^r_c\mathrm{G}^i_{j,1}, {}^r_c\,\mathrm{G}^i_{j,2}) < \lambda_{max} \tag{11}$$

The proof of Theorem 2 and Theorem 3 will be shown in Appendix. From Theorem 3, we know that formula (11) is true. Because one $CUT({}^c_r\mathrm{G}^i_{j,1}, {}^c_r\,\mathrm{G}^i_{j,2})$ corresponds to one eigenvalue of L and the smallest eigenvalue of L is 0, we know that Theorem 1 is true. Therefore, if we can get the second smallest eigenvalue and its eigenvector, we can get the minimum cut and its corresponding sub-graphs. But the calculation of eigenvalue is very time consuming. In this paper, we calculate the eigenvalues of L using Spark framework which can significantly reduce the computing time. The corresponding two parts of the cut can be gotten from the eigenvector corresponding to the second smallest eigenvalue.

Offloading scheme generation: We can get the minimum cut of each compressed sub-graph ${}^r_c\mathrm{G}^i_j$ from the eigenvector calculation. Then all compressed sub-graphs will be split into two parts. The last step is how to find the optimal allocation scheme of these parts such that the energy consumption and time consumption can be minimized (formula (6)). We do this based on the greedy strategy [4].

We design an offloading scheme generation algorithm which is shown in Algorithm 2.

We first compress all user's function data flow graph (Line 1). For each compressed sub-graph which is corresponding to one component, we find its minimum cut and split it into two parts (Line 2 to Line 6). The optimal offloading scheme will be generated based on greedy searching in all parts (Line 7 to Line 12).

IV. EXPERIMENTS

In this section, some experiments are conducted to prove the performance and accuracy of our algorithms. First, the compression algorithm is tested. Second, for the offloading scheme, three algorithms are selected separately. In addition to our algorithm, we chose the maximum flow minimum cut algorithm and the Kernighan–Lin algorithm for comparison. These two algorithms are proven as effective algorithms for graph partition and briefly introduced in following paragraph.

The Ford-fulkerson algorithm for maximum flow minimum cut and Kernighan-Lin algorithm are briefly introduced here. Ford-fulkerson algorithm which is used to solve maximum flow finding from source node s to target or sink node t. A specialized Ford-Fulkerson algorithm, also called as Edmond-Karp algorithm guarantees to find maximum flow in limited number of iterations. In Ford-Fulkerson algorithm [27], if the capacities are rational, then the algorithm does give result in limited iterations. Except for s and t, for all other nodes the incoming flow is equal to the outgoing flow. Flow at any edge is always less than the capacity of the edge. When flow f passes through the capacity c of an edge then the residual capacity becomes $c-f$. An edge with available capacity is called as augmenting path. The idea is to find augmented path from s to t in such a way that the flow is maximized.

Kernighan-Lin algorithm is mainly used for partitioning of network graph. Kernighan-Lin algorithm is a heuristic algorithm for graph partitioning problem [7]. This algorithm is a feasible algorithm for graph partition. Kernighan-Lin algorithm is used to partition a network node graph into two sets based on some constraints. One example of such constraint might be connecting edges between the two sets in which this graph is partitioned should have minimum weight. Another possible constraint can be maximizing the function gain after partitioning. Let Q be the function gain after partitioning. Which node should remain local and which should be offloaded is decided based on the arrangement producing maximum value of Q. The Kernighan-Lin algorithm has been successfully applied to the partition of graphs and we have simulated it [21].

We use **NETGEN** [18] to generate experimental data. **NETGEN** is a fast tool for randomly generating network graph based on the number of nodes, the number of edges and the weight of edges provided by users. We set the number of edges and values of weights in the graph so that the generated random graph is similar to the actual function data flow graph of mobile applications.

Experiment on graph compression algorithm: Table I reflects the result of our graph compression algorithm. The scale of the original graphs is reduced a lot. With the increase of graph size, the compression ratio also increases. When the graph node number is 5000, the number of nodes can be reduced is more than 90%.

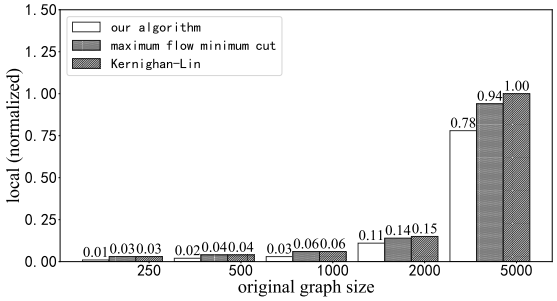

Figure 3. Local energy consumption.

Experiments on offloading scheme generation algorithm: We conduct a series of experiments on offloading scheme generation algorithms. We change the minimum cut calculation process by the above mentioned three algorithms and compare their results.

We control the number of users to one and observe the energy consumption of the mobile terminal in the result

Table I
GRAPH COMPRESSION RESULTS

Network	function number	edge number	function number after compression	edge number after compression
Network1	250	1214	39	107
Network2	500	2643	93	284
Network3	1000	4912	154	597
Network4	2000	9578	258	1220
Network5	5000	40243	489	2651

schemes. We use three algorithms to cut graphs of different sizes. As can be seen from figure 2, with the increase of the scale of the original graph, the local consumption is also increasing, because more functions will execute locally. The transmission energy consumption and the total consumption have the same trend (figure 4 and figure 5). Through comparison, our algorithm has the best performance either on local energy consumption (figure 3) or on transmission energy consumption (figure 4). Therefore, its total energy consumption is also the least (figure 5).

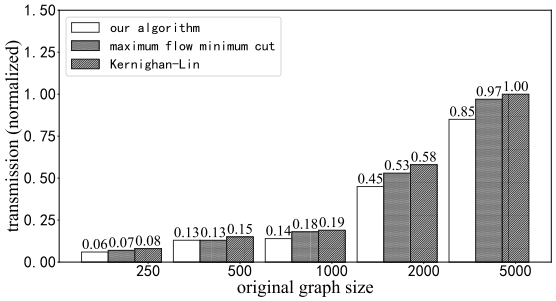

Figure 4. Transmission energy consumption.

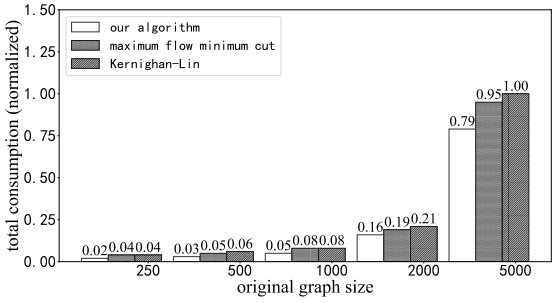

Figure 5. Total energy consumption.

We set the function number of graph to 1000 and observe the change of local energy consumption (figure 6), transmission energy consumption (figure 7) and the total energy consumption (figure 8) by increasing the number of users. The results are consistent with the single user situation

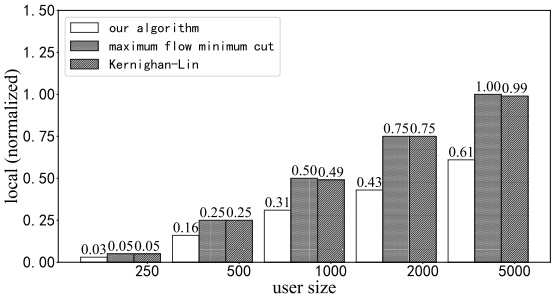

Figure 6. Local energy consumption under multi-user conditions.

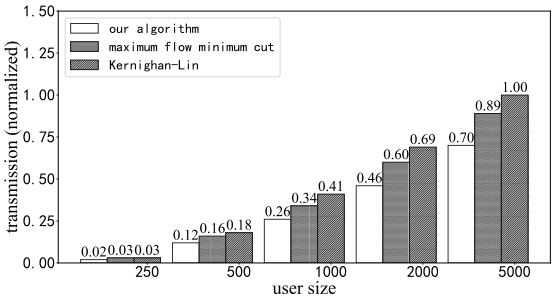

Figure 7. Transmission energy consumption under multi-user conditions.

We continuously increase the scale of graphs to observe the running time of these three algorithms. As can be seen from figure 9, the running time of our algorithm without using Spark framework is significantly greater than that of the other two algorithms when the scale of the graph keep increasing. Most of the running time is wasted on lots of matrix multiplications about the graph spectrum calculation. When we use Spark to do the matrix multiplications [25], the running time is close to the other two algorithms.

V. CONCLUSIONS

Computation offloading is very promising technique for enhancing the capabilities of mobile devices by transferring offloadable applications or functions to edge servers. In this paper, Computation is tackled using algorithms based on the label propagation theory and graph spectral theory. For successful offloading of partial functions, we have identified

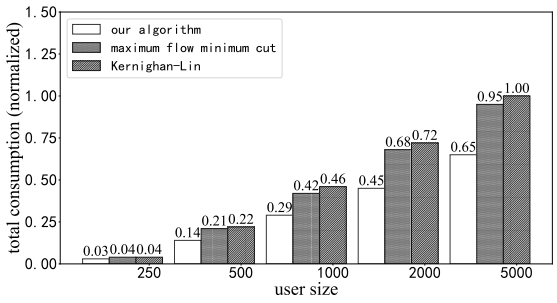

Figure 8. Total energy consumption under multi-user conditions.

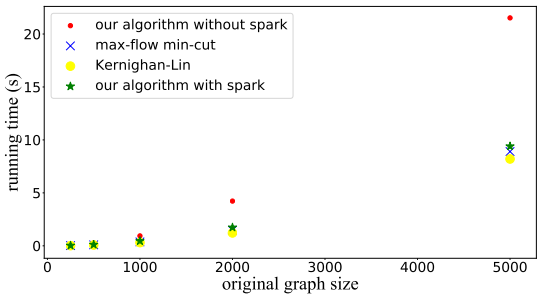

Figure 9. Execution time under multi-user condition.

the offloadable and unoffloadable part of function first. Then the offloading problem is represented in graphical form where nodes represent functions and edges represent the communication between functions. In order to achieve the optimal offloading scheme, the original function data flow graph is compressed and the problem is transformed into optimal cut searching problem. We design a series of label rule, propagation rule and compression rule for the label propagation based graph compression algorithm. We use graph spectrum theory to solve the optimal cut problem of the graph. The optimal offloading scheme is generated by greedy searching. Experiments are carried out using Spark technology to accelerate the speed and improve overall performance of the propsed algorithm. Evaluation results indicate that our algorithm also produces better results in reducing transmission consumption and energy consumption during offloading. In future, we plan to explore different ways to reduce the computational complexity of our algorithm.

VI. Acknowledgements

This work is partly supported by National Science Foundation under Grant 1747818, and the Fundamental Research Funds for Central Universities (JB161004).

References

[1] Khadija Akherfi, Micheal Gerndt, and Hamid Harroud. “Mobile cloud computing for computation offloading: Issues and challenges”. In: *Applied Computing Informatics* 14.1 (2016), S2210832716300400.

[2] Eduardo Cuervo et al. “MAUI: Making smartphones last longer with code offload”. In: *International Conference on Mobile Systems*. 2010.

[3] Hoang T Dinh et al. “A survey of mobile cloud computing: architecture, applications, and approaches”. In: *Wireless communications and mobile computing* 13.18 (2013), pp. 1587–1611.

[4] Luobing Dong, Qiumin Guo, and Weili Wu. “Speech corpora subset selection based on time-continuous utterances features”. In: *Journal of Combinatorial Optimization* (2018), pp. 1–12.

[5] J. A. Gutiérrez-Pérez et al. “Application of graph-spectral methods in the vulnerability assessment of water supply networks ”. In: *Mathematical Computer Modelling* 57.7-8 (2013), pp. 1853–1859.

[6] Yun Chao Hu et al. “Mobile edge computing—A key technology towards 5G”. In: *ETSI white paper* 11.11 (2015), pp. 1–16.

[7] Brian W Kernighan and Shen Lin. “An efficient heuristic procedure for partitioning graphs”. In: *The Bell system technical journal* 49.2 (1970), pp. 291–307.

[8] Watcharachai Kongsiriwattana et al. “Smart-phone battery-life short-fall in disaster response: Quantifying the gap”. In: *Global Humanitarian Technology Conference*. 2017.

[9] Nicholas D Lane et al. “Deepx: A software accelerator for low-power deep learning inference on mobile devices”. In: *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press. 2016, p. 23.

[10] Juan Liu et al. “Delay-Optimal Computation Task Scheduling for Mobile-Edge Computing Systems”. In: *IEEE International Symposium on Information Theory*. 2016.

[11] Yanchen Liu and Myung J. Lee. “An Effective Dynamic Programming Offloading Algorithm in Mobile Cloud Computing System”. In: *Wireless Communications Networking Conference*. 2014.

[12] Michael W Mahoney, Lorenzo Orecchia, and Nisheeth K Vishnoi. “A local spectral method for graphs: with applications to improving graph partitions and exploring data graphs locally”. In: (2012).

[13] Yuyi Mao et al. “A Survey on Mobile Edge Computing: The Communication Perspective”. In: *IEEE Communications Surveys Tutorials* PP.99 (2017), pp. 1–1.

[14] S. Ou, K. Yang, and A. Liotta. "An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems". In: *IEEE International Conference on Pervasive Computing Communications*. 2006.

[15] Aditya Rajgarhia and Ashish Gehani. "Performance and extension of user space file systems". In: *Acm Symposium on Applied Computing*. 2010.

[16] Heungsoon Rim et al. "Transparent Method Offloading for Slim Execution". In: *International Symposium on Wireless Pervasive Computing*. 2006.

[17] Stefania Sardellitti, Gesualdo Scutari, and Sergio Barbarossa. "Joint optimization of radio and computational resources for multicell mobile-edge computing". In: *IEEE Transactions on Signal and Information Processing over Networks* 1.2 (2015), pp. 89–103.

[18] Joachim Schöberl. "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules". In: *Computing Visualization in Science* 1.1 (1997), pp. 41–52.

[19] Henry Smith. "A Survey of Computation Offloading for Mobile Systems". In: *Mobile Networks Applications* 18.1 (2013), pp. 129–140.

[20] Raja. Vallee-Rai. "Soot : a java bytecode optimization framework". In: *Proceedings of Cascon* (2010), pp. 214–224.

[21] Raf Van Driessche and Robert U Piessens. "Parallel graph partitioning with the Kernighan-Lin heuristic". In: (1994).

[22] Changjiu Xian, Yung Hsiang Lu, and Zhiyuan Li. *Adaptive computation offloading for energy conservation on battery-powered systems*. 2007.

[23] Kun Yang, Shumao Ou, and Hsiao Hwa Chen. "On effective offloading services for resource-constrained mobile devices running heavier mobile Internet applications". In: *Communications Magazine IEEE* 46.1 (2008), pp. 56–63.

[24] Shanhe Yi, Cheng Li, and Qun 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.

[25] Reza Bosagh Zadeh et al. "Matrix Computations and Optimization in Apache Spark". In: *Acm Sigkdd International Conference on Knowledge Discovery Data Mining*. 2016.

[26] Xiaojin Zhu and Zoubin Ghahramani. "Learning from Labeled and Unlabeled Data". In: *Tech Report* 3175.2004 (2002), pp. 237–244.

[27] Uri Zwick. "The smallest networks on which the Ford-Fulkerson maximum flow procedure may fail to terminate". In: *Theoretical Computer Science* 148.1 (1995), pp. 165–170.

APPENDIX A.
PROOF OF THEOREM 2

Proof:

We assume that L represent the Laplace matrix of ${}^{c}_{r}\mathrm{G}^{i}_{j}$.

$q = (q_1, q_2, ..., q_n)^T$ is a vector. For any element q_i in q, $q_i = \begin{cases} d_1, & v^i_a \in G^i_{j,1} \\ d_2, & v^i_a \in G^i_{j,2} \end{cases}$. v^i_a represents the ath node of ${}^{c}_{r}\mathrm{G}^{i}_{j}$. We can get:

$$CUT({}^{c}_{r}\mathrm{G}^{i}_{j,1}, {}^{c}_{r}\mathrm{G}^{i}_{j,2}) = \frac{\sum_{a=1}^{n}\sum_{b=1}^{n} s(v^i_a, v^i_b)(q_a - q_b)^2}{2(d_1 - d_2)^2} = \frac{q^T L q}{(d_1 - d_2)^2}$$

APPENDIX B.
PROOF OF THEOREM 3

Proof:

From Theorem 2, we know that:

$$CUT({}^{c}_{r}\mathrm{G}^{i}_{j,1}, {}^{c}_{r}\mathrm{G}^{i}_{j,2}) = \frac{q^T L q}{(d_1 - d_2)^2}$$

so $CUT({}^{c}_{r}\mathrm{G}^{i}_{j,1}, {}^{c}_{r}\mathrm{G}^{i}_{j,2})$ has the same extreme points to $\frac{q^T L q}{(d_1 - d_2)^2}$.

then $CUT({}^{c}_{r}\mathrm{G}^{i}_{j,1}, {}^{c}_{r}\mathrm{G}^{i}_{j,2})$ has the same extreme points to $q^T L q$.

Assume that γ is a constant, and $\gamma \neq 0$. We know that

$$(\gamma q)^T L(\gamma q) = \gamma^2 q^T L q$$

so $(\gamma q)^T L(\gamma q)$ has the same extreme points to $q^T L q$.

Therefore, we can assume that $\|q\| = q^T q = 1$. When this condition is not true, the following result is still true.

From the method of Lagrange multipliers, we can get

$\iota(q) = q^T Lq - \lambda(q^T q - 1)$

so $\frac{d\iota(q)}{dq} = 0$

$<\!\!=\!\!>\ 2q^T L - 2\lambda q^T = 0$

$<\!\!=\!\!>\ 2Lq - 2\lambda q = 0$

$<\!\!=\!\!>\ 2Lq - 2\lambda q = 0$

$<\!\!=\!\!>\ Lq = \lambda q.$

Therefore, Theorem 3 is true.