An Efficient Application Partitioning Algorithm in Mobile Environments

Huaming Wu, *Member, IEEE*, William J. Knottenbelt and Katinka Wolter

Abstract—Application partitioning that splits the executions into local and remote parts, plays a critical role in high-performance mobile offloading systems. Optimal partitioning will allow mobile devices to obtain the highest benefit from Mobile Cloud Computing (MCC) or Mobile Edge Computing (MEC). Due to unstable resources in the wireless network (network disconnection, bandwidth fluctuation, network latency, etc.) and at the service nodes (different speeds of mobile devices and cloud/edge servers, memory, etc.), static partitioning solutions with fixed bandwidth and speed assumptions are unsuitable for offloading systems. In this paper, we study how to dynamically partition a given application effectively into local and remote parts while reducing the total cost to the degree possible. For general tasks (represented in arbitrary topological consumption graphs), we propose a Min-Cost Offloading Partitioning (MCOP) algorithm that aims at finding the optimal partitioning plan (i.e. to determine which portions of the application must run on the mobile device and which portions on cloud/edge servers) under different cost models and mobile environments. Simulation results show that the MCOP algorithm provides a stable method with low time complexity which significantly reduces execution time and energy consumption by optimally distributing tasks between mobile devices and servers, besides it adapts well to mobile environmental changes.

Index Terms—Mobile cloud computing, mobile edge computing, communication networks, offloading, application partitioning.

1 Introduction

Along with the development of Mobile Cloud Computing (MCC) and Mobile Edge Computing (MEC) strategies computation offloading is becoming a promising method to reduce task execution time and prolong the battery life of mobile devices. When comparing both, MEC can offer significantly lower latency but has less computational and storage resources than MCC [1]. The main idea of computation offloading is to migrate heavy computation from mobile devices to resourceful cloud/edge servers from where the result are then received via wireless networks. Offloading is an effective way to overcome constraints in resources and functionalities of mobile devices since it can release them from intensive processing [2].

Offloading all computation components of an application to the remote cloud or nearby edge server is not always necessary or effective. Especially for some complex applications (e.g., QR-code recognition, online social applications, health monitoring using body sensor networks) that can be divided into a set of independent parts, a mobile device should judiciously determine whether to offload computation and which portion of the application should be offloaded to the server. Offloading decisions must be taken for all parts, and the decision made for one part may depend on the one for other parts. As mobile computing increasingly interacts with the cloud, a number of approaches have been proposed, e.g., MAUI [3] and CloneCloud [4], both systems that offload some parts of the mobile application execution to the cloud. To achieve good performance, they particularly focus on a specific application partitioning problem, i.e., to decide which parts of an application should be offloaded to powerful servers in a remote cloud and which parts should be executed locally on the mobile device such that the total execution cost is minimized. Through partitioning, a mobile device can benefit most from offloading. Thus, partitioning algorithms play a critical role in high-performance offloading systems.

The main costs for mobile offloading systems are the computational cost for local and remote execution, respectively, and the communication cost due to the extra communication between the mobile device and the cloud/edge server. Program execution can naturally be described as a graph in which vertices represent computation that are labeled with the computation costs and edges reflect the sequence of computation labeled with communication costs [5] when computation is carried out in different places. By partitioning the vertices of a graph, the calculation can be divided among processors of local mobile devices and servers. Traditional graph partitioning algorithms (e.g., [6], [7], [8]) cannot be applied directly to the mobile offloading systems, because they only consider the weights on the edges of the graph, neglecting the weight of each node. Our research is situated in the context of resource-constrained mobile devices, in which there are often multi-objective partitioning cost functions subject to variable vertex cost, such as minimizing the total response time or energy consumption on mobile devices by offloading partial workloads to a server through links with fluctuating reliability.

- *H. Wu is with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China (email: whming@tju.edu.cn). W.J. Knottenbelt is with the Department of Computing, Imperial College London, London SW7 2AZ, United Kingdom (email: wjk@doc.ic.ac.uk). K. Wolter is with the Institut für Informatik, Freie Universität Berlin, Berlin 14195, Germany (email: katinka.wolter@fu-berlin.de).*
- *This work was supported by the the National Natural Science Foundation of China (Grant Number: 61801325), the Natural Science Foundation of Tianjin City (Grant Number: 18JCQNJC00600) and the Huawei Innovation Research Program (Grant Number: HIRPO2017050307).*

With respect to the conference paper [9], the main contributions of this paper can be concluded as follows:

- We develop a dynamic version of the partitioning algorithm, that is able to determine the optimal division into the classes of jobs for remote and for local execution fast and repeatedly whenever system parameters change [10]. We further apply the proposed MCOP algorithm to general topology either for MCC-based offloading or MEC-based offloading.
- Offloading decisions are based on the resource consumption (CPU speed, network bandwidth, transmission data size and speed of the cloud/edge server) [11]. We construct a weighted resource consumption graph (WCG) and further derive a novel *min-cost offloading partitioning (MCOP)* algorithm designed especially for mobile offloading systems.
- We include the developed algorithms into a workflow of an environment-adaptive application partitioning processes which considers network reliability, real-time adaptability, and partitioning efficiency.
- We consider profiling for adaptive partitioning and use our implementation for experiments. We provide three profilers, a *program profiler*, a *network profiler*, and an *energy profiler* to collect information about the device and network characteristics, which are critical parts of the partitioning algorithm.

The remainder of this paper is organized as follows. We review related work in Section 2. Section 4 explores the partitioning challenges and processes, and describes profilers that are used for information collection. Section 3 introduces the partitioning models such as topology, optimization and partitioning cost models. An optimal partitioning algorithm for arbitrary topology is proposed and studied in Section 5. Section 6 gives some evaluation and simulation results. Finally, the paper is summarized in Section 7.

2 Related Work

Offloading becomes an attractive solution for meeting response time requirements and extending battery lifetime on mobile systems as applications become increasingly complex [12]. Karthik *et al*. [13] argued that offloading could potentially save energy and reduce execution time for mobile users, but not all applications are energy-efficient and time-saving when they are migrated to the cloud. It depends on whether the computation cost saved due to offloading outperforms the extra communication cost. A large amount of communication combined with a small amount of computation should preferably be performed locally on the mobile device, while a small amount of communication with a large amount of computation should preferably be executed remotely.

Many research efforts have been devoted to computation partitioning in mobile computing, in order to shorten response time or to save energy.

Compared with offloading a whole application to the cloud, a partitioning scheme is able to achieve a fine granularity for computation offloading [14]. A partitioning algorithm as introduced in [15] aims at reducing the response time of tasks on mobile devices. It finds the offloading and integration points on a sequence of calls by depth-first search and a linear time search scheme. It can achieve low user-perceived latency while largely reducing the partitioning computation on cloud. Some application partitioning solutions like [16], [17] heavily depend upon programmers and middleware to partition the applications, which limits their applicability. Hence, automatic application partitioning like [18], [19] attracts more attention. The offloading inference engine proposed in [18] can adaptively make decisions at runtime, dynamically partition an application and offload parts of the application execution to a powerful nearby surrogate.

Partitioning technologies were adopted to identify offloaded parts for energy saving [3], [20], [21]. The energy cost of each function of the application was profiled and according to the profiling result a cost graph was constructed, in which each node represents a function to be performed and each edge indicated the data to be transmitted. Finally, the server parts were executed on remote servers for reducing the energy consumption. CloneCloud [4] uses a combination of static analysis and dynamic profiling to partition applications automatically at a fine granularity while optimizing the energy usage for a target computation and communication environment. However, this approach only considers limited input/environmental conditions in the offline pre-processing and needs to be bootstrapped for every new application built [22]. Due to frequent bandwidth fluctuations in the mobile environment, static application partitioning like [23], [24] cannot work well on mobile platforms. The availability of resources may change at the service nodes (available CPU power, memory, file cache, etc.) and at the wireless network (bandwidth, network latency, etc.) [25]. Thus, optimal partitioning decisions should be made dynamically at runtime to adapt to different operating conditions. A framework was designed in [26] for runtime computation repartitioning in dynamic mobile cloud environments to solve the performance degradation issue arising from dynamic network and device status, however it neglects the energy consumption of the mobile device. In [27], the offloading operation is modeled via a cost graph, where finding the best solution for offloading is equivalent to finding the constrained shortest path in this linear graph. Besides considering the linear call graph for the program, a mobile application is modeled as a general topology in [28], consisting of a set of fine-grained tasks. Each task within the application can be either executed on the mobile device or on the cloud. They tried to find the optimized path that can obtain minimum energy or delay cost.

This work was motivated by the above work and we investigate the partitioning problem in a dynamic environment, in which the network has disconnection and bandwidth fluctuation, aiming at the different objectives, e.g., minimum response time, minimum energy consumption, and minimum of the weighted sum of response time and energy. We explicitly considered the mobile nature of both user and application behaviors and consider how dynamic partitioning can address these heterogeneity problems by using the bandwidth as a variable. Thus, we have greatly extended prior work by considering dynamic partitioning of applications between weak devices and cloud/edge servers, in order to better support applications running on diverse

devices in different environments.

3 System Model

In this section, we will discuss the assumptions made in this paper, how the weighted consumption graphs (WCGs) for different types of applications are constructed and how cost models are defined.

3.1 Classification of Topologies

Flexible partitioning granularity-based applications are not limited to a specific form. Previous work considers application partitioning at different levels of granularity: classes [29], objects [30], methods [3], components [14], [31], and threads [4]. Without loss of generality, we refer to application tasks in this paper. Application developers can choose the appropriate partition granularity according to their different applications.

The construction of WCGs is essential for application partitioning. A mobile application can be represented as a list of fine-grained tasks, combined to build different topologies as depicted in Fig. 1, where each node denotes an application task, executed either locally at the mobile side or offloaded onto the server side for remote execution, and each edge denotes the data dependency between tasks [28]. Input/output data of an application that is transferred between the mobile device and the server includes relevant data and code (e.g., application data like image/video data, mobile system settings, parameters, program codes, intermediate states and return values of method invocations) involved in the task.

(a) *Only one active node*: representing an entire application (without partitioning). Such a topology is often adopted by previous full offloading schemes such as [4], [32], [33], [34], which can also be viewed as an example of *Software as a Service*. In this case the whole application is migrated to a remote server involving complete transfer of code and program state to the server [35]. The main drawback of this solution includes inflexibility and coarse granularity.
(b) *Linear topology*: representing a sequential list of fine-grained tasks [15]. Each task is sequentially executed, with output data generated by one task as the input of the next one [36].
(c) *Loop-based topology*: a loop-based application is one in which most of the functionality is given by iterating an execution loop, such as all the online social applications, which we model with a graph that consists of a cycle [37].
(d) *Tree-based topology*: representing a tree-based hierarchy of tasks [35]. The node at the top of the tree is the application entry node (i.e., the main module).
(e) *Mesh-based topology*: representing a lattice-based topology of tasks, e.g., a Java example of face recognition as depicted in [30].

When compared with the scheme that offloads the whole application (i.e., Fig. 1(a)) to the server, an application partitioning scheme is able to achieve a fine granularity for computation offloading. This is done by partitioning a topological consumption graph (CG) between local and remote execution. Different partitions can lead to different costs, and the total cost incurred due to offloading depends on multiple factors, such as device platforms, networks, clouds, and workloads. Therefore, the application may have different optimal partitions for different mobile environments and workloads.

3.2 Weighted Consumption Graphs

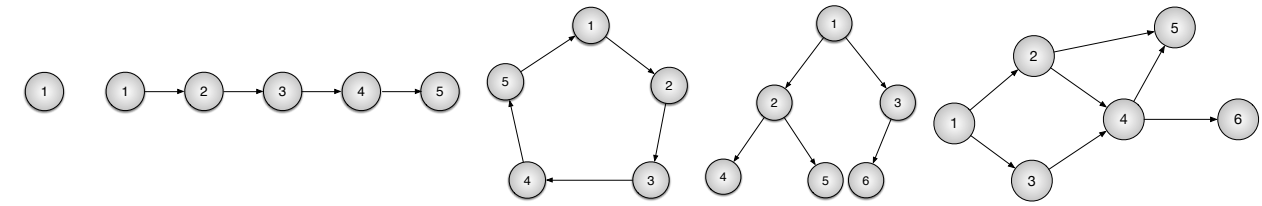
There are two types of cost in offloading systems: one is computational cost of running an application tasks locally or remotely (including memory cost, processing time cost etc.) and the other is communication cost for the application tasks' interaction (associated with movement of data and requisite messages). Even the same task can have different costs on the mobile device and servers in terms of execution time and energy consumption. As cloud/edge servers usually process much faster than mobile devices having a powerful configuration, energy can be saved and performance can be improved when offloading part of the computation to remote servers [38]. However, when vertices are assigned to different sides, the interaction between them leads to extra communication costs. Therefore, we try to find the optimal assignment of vertices for graph partitioning and computation offloading by trading off the computational cost against the communication cost.

Call graphs are widely used to describe data dependencies within a computation, where each vertex represents a task and each edge represents the calling relationship from the caller to the callee. Figure 2(a) shows a CG example consisting of six tasks [16]. The computation costs are represented by vertices, while the communication costs are expressed by edges. We depict the dependency of application tasks and their corresponding costs as a directed acyclic graph $G = (V, E)$, where the set of vertices $V = (v_1, v_2, \cdots, v_N)$ denotes N application tasks and an edge $e(v_i, v_j) \in E$ represents the frequency of invocation and data access between nodes v_i and v_j, where vertices v_i and v_j are neighbors. Each task v_i is characterized by five parameters:

- *type*: offloadable or unoffloadable task.
- m_i: the memory consumption of v_i on a mobile device platform,
- c_i: the size of the compiled code of v_i,
- in_{ij}: the data size of input from v_i to v_j,
- out_{ji}: the data size of output from v_j to v_i.

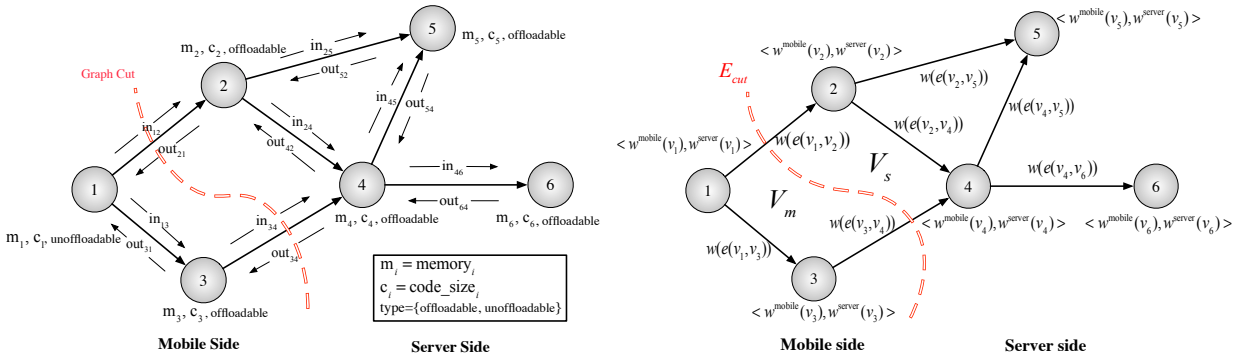
We further construct a WCG as depicted in Fig. 2(b) depending on profiling techniques. Each vertex $v \in V$ is annotated with two cost weights: $\boldsymbol{w}(v) =< w^{\text{mobile}}(v), w^{\text{server}}(v) >$, where $w^{\text{mobile}}(v)$ and $w^{\text{server}}(v)$ represent the computation cost of executing task v locally on the mobile device and remotely on the server, respectively. Each vertex is assigned one of the values in the tuple depending on the partitioning result of the resulting application graph [39]. The edge set $E \subset V \times V$ represents the communication cost among tasks. The weight of an edge $w(e(v_i, v_j))$ is denoted as:

$$w(e(v_i, v_j)) = \frac{in_{ij}}{B_{\text{upload}}} + \frac{out_{ij}}{B_{\text{download}}}, \tag{1}$$



(a) One (b) Linear (c) Loop (d) Tree (e) Mesh

Fig. 1: Task-flow graphs for different topologies



(a) CG (b) WCG

Fig. 2: Consumption Graph (CG) and Weighted Consumption Graph (WCG).

which is the communication cost of transferring the input and return states when the tasks v_i and v_j are executed on different sides. The communication cost closely depends on the network bandwidth (upload B_{upload} and download B_{download}) and reliability as well as the amount of transferred data.

A candidate offloading decision is described by one cut in the WCG, which separates the vertices into two disjoint sets, one representing tasks that are executed on the mobile device and the other one implying tasks that are offloaded to the remote server [40]. Hence, taking the optimal offloading decision is equivalent to partitioning the WCG such that an objective function is minimized.

The red dotted line in Fig. 2(b) is one possible partitioning cut, indicating the partitioning of computational workload in the application between the mobile side and the server side. V_m and V_s are sets of vertices, where V_m is the mobile set in which tasks are executed locally at the mobile side and V_s is the server set in which tasks are directly offloaded to the server. We have $V_m \cap V_s = \emptyset$ and $V_m \cup V_s = V$. Further, E_{cut} is the edge set in which the graph is cut into two parts.

3.3 Cost Models

Mobile application partitioning aims at finding the optimal partitioning solution that leads to the minimum execution cost in order to make the best tradeoff between time/energy savings and transmission costs/delay.

The optimal partitioning decision depends on user requirements/expectations, device information, network bandwidth, and the application itself. Device information includes the execution speed of the device and the workloads on it when the application is launched. If the device computes very slowly and the aim is to reduce execution time, it is better to offload more computation to the cloud/edge server. However, network bandwidth affects data transmission for remote execution. If the bandwidth is very high, the cost in terms of data transmission will be low. In this case, it is better to offload more computation to the server.

The partitioning decision is made based on the cost estimation (computational and communication costs) before the program execution. On the basis of Fig. 2(b), we can formulate the partitioning problem as:

$$C_{\text{total}} = \underbrace{\sum_{v \in V} I_v \cdot w^{\text{mobile}}(v)}_{\text{mobile}} + \underbrace{\sum_{v \in V} (1 - I_v) \cdot w^{\text{server}}(v)}_{\text{server}} + \underbrace{\sum_{e(v_i, v_j) \in E} I_e \cdot w(e(v_i, v_j))}_{\text{communication}}, \quad (2)$$

where the total cost is the sum of computational costs (mobile and server) and communication costs of cut affected edges.

The server node and the mobile device node must belong to different partitions. One possible solution for this partitioning problem will give us an arbitrary tuple of partitions from the vertices set $< V_m, V_s >$ and the cut of edge set E_{cut} in the following way:

$$I_v = \begin{cases} 1, & \text{if } v \in V_m \\ 0, & \text{if } v \in V_s \end{cases} \quad \text{and} \quad I_e = \begin{cases} 1, & \text{if } e \in E_{\text{cut}} \\ 0, & \text{if } e \notin E_{\text{cut}} \end{cases}. \quad (3)$$

We seek to find an optimal cut: $\boldsymbol{I}_{\text{min}} = \{I_v, I_e | I_v, I_e \in \{0, 1\}\}$ in the WCG such that some application tasks are executed on the mobile side and the remaining ones on the cloud side, while satisfying the general goal of a partition: $\boldsymbol{I}_{\text{min}} = \arg\min_{\boldsymbol{I}} C_{\text{total}}(\boldsymbol{I})$. The optimal cut maximizes or minimizes an objective function and while satisfying the

resource constraints of a mobile device. The dynamic execution configuration of an elastic application can be decided based on different objectives with respect to response time and energy consumption which have been saved for parallel use. Partitioning is performed only when it is beneficial, but different partitionings are derived beforehand. For the overall cost estimate the cost estimation of running each application task on the mobile device and edge/cloud server is needed. Offloading makes sense only if the speedup of the server covers the extra communication costs.

The communication time and energy costs for the mobile device will vary according to the amount of data to be transmitted and the wireless network conditions. According to (2) the dynamic execution configuration of an elastic application can be decided based on selecting on out of different objectives with respect to response time and energy consumption which have been saved. The offloading objectives of a task may change due to a change in environmental conditions.

3.3.1 Minimum Response Time

The communication cost depends on the size of data transfer and the network bandwidth, while the computation time has an impact on the computation cost. If the minimum response time is selected as objective function, we can calculate the total time spent due to offloading as:

$$T_{\text{total}}(\boldsymbol{I}) = \underbrace{\sum_{v\in V} I_v \cdot T_v^m}_{\text{mobile}} + \underbrace{\sum_{v\in V} (1-I_v)\cdot T_v^s}_{\text{server}} + \underbrace{\sum_{e\in E} I_e \cdot T_e^{tr}}_{\text{communication}}, \quad (4)$$

where $T_v^m = F \cdot T_v^s$ is the computation time of task v on the mobile device when it is executed locally; F is the speedup factor, which is the ratio of the execution speed of the server with that of the mobile device (or the inverse ratio of their mean task completion times). Usually, the computation capacity of a server is higher than that of a mobile device and we normally have $F > 1$; T_v^s is the computation time of task v on the cloud/edge server when it is offloaded; $T_e^{tr} = D_e^{tr}/B$ is the communication time between the mobile device and the cloud; D_e^{tr} is the amount of data that is transmitted and received; finally, B is the current wireless bandwidth weighed with the reliability of the network.

In this scenario, the offloading decision engine then selects the best partitioning candidate that minimizes the total response time. The aim of this cost model is to find the optimal application partitioning: $\boldsymbol{I}_{\text{min}} = \{I_v, I_e | I_v, I_e \in \{0,1\}\}$, which satisfies $\boldsymbol{I}_{\text{min}} = \arg\min_{\boldsymbol{I}} T_{\text{total}}(\boldsymbol{I})$. For a given application and a mobile device the optimal partitioning result also changes according to different wireless network bandwidth and speedup factor of the server (i.e. different relative mean task completion time of mobile device and server).

The amount of response time which is saved by the partitioning scheme compared to the computation without offloading is calculated as:

$$T_{\text{save}}(\boldsymbol{I}) = \frac{T_{\text{mobile}} - T_{\text{total}}(\boldsymbol{I})}{T_{\text{mobile}}} \cdot 100\%, \quad (5)$$

where $T_{\text{mobile}} = \sum_{v\in V} T_v^m$ is the local time cost when all the application tasks are executed locally on the mobile device.

3.3.2 Minimum Energy Consumption

If the minimum energy consumption is chosen as the objective function, we can calculate the total energy consumed with offloading as:

$$E_{\text{total}}(\boldsymbol{I}) = \underbrace{\sum_{v\in V} I_v \cdot E_v^m}_{\text{mobile}} + \underbrace{\sum_{v\in V} (1-I_v)\cdot E_v^i}_{\text{idle}} + \underbrace{\sum_{e\in E} I_e \cdot E_e^{tr}}_{\text{communication}}, \quad (6)$$

where $E_v^m = p_m \cdot T_v^m$ is the energy consumed by task v on the mobile device when it is executed locally; $E_v^i = p_i \cdot T_v^c$ is the energy consumed by task v on the mobile device when it is offloaded to the server; $E_e = p_{tr} \cdot T_e^{tr}$ is the energy spent on the communication between the mobile device and the server; p_m, p_i and p_{tr} are the compute power of the mobile device for computing while being idle and for data transfer, respectively.

In this scenario the offloading decision engine then selects the best partitioning plan that minimizes the partitioning cost of energy. The aim is to find the optimal application partitioning: $\boldsymbol{I}_{\text{min}} = \{I_v, I_e | I_v, I_e \in \{0,1\}\}$, which satisfies: $\boldsymbol{I}_{\text{min}} = \arg\min_{\boldsymbol{I}} E_{\text{total}}(\boldsymbol{I})$.

The saved energy when compared to the scheme without offloading is:

$$E_{\text{save}}(\boldsymbol{I}) = \frac{E_{\text{mobile}} - E_{\text{total}}(\boldsymbol{I})}{E_{\text{mobile}}} \cdot 100\%, \quad (7)$$

where $E_{\text{mobile}} = \sum_{v\in V} E_v^m$ is the local energy cost when all tasks are executed locally on the mobile device.

3.3.3 Minimum of the Weighted Sum of Time and Energy

If we combine both the response time and energy consumption [41], we can design the cost model for partitioning as follows:

$$W_{\text{total}}(\boldsymbol{I}) = \omega \cdot \frac{T_{\text{total}}(\boldsymbol{I})}{T_{\text{mobile}}} + (1-\omega) \cdot \frac{E_{\text{total}}(\boldsymbol{I})}{E_{\text{mobile}}}, \quad (8)$$

where $0 \le \omega \le 1$ is a weighting parameter used to share the relative importance between the response time and energy consumption. Large ω favors response time while small ω favors energy consumption [42]. In some special cases performance can be traded for power consumption and vice versa [43], therefore we can use the ω parameter to express preferences for different applications in such special cases. $T_{\text{total}}(\boldsymbol{I})$ and $E_{\text{total}}(\boldsymbol{I})$ are the response time and energy consumption with the partitioning solution $\boldsymbol{I}$, respectively. To eliminate the impact of different scales of time and energy, they are divided by the local costs. If $T_{\text{total}}(\boldsymbol{I})/T_{\text{mobile}}$ is less than 1, the partitioning will increase the power consumption of the application. Similarly, if $E_{\text{total}}(\boldsymbol{I})/E_{\text{mobile}}$ is less than 1, it will reduce the performance of the application, i.e. increase the response time of the application.

In this scenario, the offloading decision engine then selects the best partition plan that minimizes the partitioning cost of weighted sum of time and energy. Its aim is to find the optimal application partitioning: $\boldsymbol{I}_{\text{min}} = \{I_v, I_e | I_v, I_e \in \{0,1\}\}$, while satisfying: $\boldsymbol{I}_{\text{min}} = \arg\min_{\boldsymbol{I}} W_{\text{total}}(\boldsymbol{I})$.

The saved weighted sum of time and energy in the partitioning scheme compared to the scheme without offloading is calculated as:

$$W_{\text{save}}(\boldsymbol{I}) = \omega \cdot \frac{T_{\text{mobile}} - T_{\text{total}}(\boldsymbol{I})}{T_{\text{mobile}}} + (1-\omega) \cdot \frac{E_{\text{mobile}} - E_{\text{total}}(\boldsymbol{I})}{E_{\text{mobile}}}. \quad (9)$$

4 Partitioning Problems

Application partitioning is very important for designing an adaptive, cost-effective, and efficient offloading system. Some critical issues concerning the partitioning problem include:

- **Weighting**: when choosing an application task to offload, we need to scale the weights of each application task regarding its resource utilization, such as memory, processing time, and bandwidth utilization [44]. The weights can vary for different mobile devices and in different execution environments. A communication overhead is introduced by the remote communication between a mobile device and a cloud/edge server.
- **Real-Time Adaptability**: partitioning algorithms should be adaptive to network and device changes. For example, an optimal partition for a high-bandwidth low-latency network and low-capacity client might not be a good partition for a high-capacity client with a bad network connection. Since the network condition is only measurable at runtime, the partitioning algorithm should be a real-time online process [15].
- **Partitioning Efficiency**: making partitioning decisions for simple applications (e.g., an alarm clock) at real-time is not difficult, but for some complex applications (e.g., speech/face recognition) which contain a large number of methods [15], a highly efficient algorithm is required to perform real-time partitioning.

4.1 Classification of Application Tasks

Different applications emerge in a mobile device according to some process and each consists of several tasks. Since not all the application tasks are suitable for remote execution, they need to be weighed and distinguished as:

- **Unoffloadable Tasks**: some tasks should be unconditionally executed locally on the mobile device, either because transferring relevant information would take tremendous time and energy or because these tasks must access local components (e.g., camera, GPS, user interface, accelerometer or other sensors) [3]. Tasks that might cause security issues when executed in a different place should also not be offloaded (e.g., e-commerce). Local processing consumes the battery power of the mobile device, but there are no communication costs or communication delays [45] involved.
- **Offloadable Tasks**: some application components are flexible tasks that can be handled either locally on the processor of the mobile device, or on the remote cloud server or nearby edge server. Many tasks fall into this category, and the offloading decision depends on whether the communication costs outweigh the difference between local and remote costs or not [12].

For this work we assume that tasks have been annotated with their type by the programmer. For unoffloadable components non offloading decisions must be taken. However, for offloadable ones, since offloading all the application tasks to the cloud/edge server is not necessary or effective under all circumstances, it is worth considering what should be executed locally on the mobile device and what should be offloaded to the server for remote execution. This decision is taken based on available networks, network response time or energy consumption of the mobile device versus the remote server. The mobile device has to take the offloading decision based on the result of a dynamic optimization problem.

4.2 Profiling

Building the WCG (the weighted call graph) is actually the most critical part of the whole technique. It closely depends on profiling, i.e., the process of gathering the information required to make good offloading decisions. Such information may consist of the computation and communication costs of the execution units (program profiler), the network status (network profiler), and the mobile device specific characteristics such as energy consumption (energy profiler). Profilers are needed to collect information about the device and network characteristics, which is a critical part of the partitioning algorithm: the more accurate and lightweight they are, the more correct decisions can be made, and the lower overhead is introduced [22].

4.2.1 Program Profiler

A program profiler (static or dynamic) collects characteristics of applications, e.g., the execution time, the memory usage and the size of data.

Static analysis obtains the control flow graph of an application by analyzing the bytecode with nodes representing objects and edges representing relations between objects. We can get all the objects and the relations between them based on method invocations by traversing the graph. Many tools and frameworks have been developed to generate the call graph of a given application, e.g., Spark [46], Cgc [7], and Soot [47].

Dynamic profiling is adopted to obtain weights of the nodes and edges. Since approximate information exists on the execution time per bytecode instruction of a Java program the execution time of objects can be estimated by the counting the corresponding bytecode instructions [48]. Data transmission between tasks includes parameters and return values of method invocations. Combining Java bytecode rewriting with pretreatment information, we can estimate the execution time for each task (node weight) and the transmission time for each invocation (edge weight). These weights can be dynamically updated according to the varying processing capability of the cloud/edge server and the wireless bandwidth.

4.2.2 Network Profiler

A network profiler collects information about the status and bandwidth of the available wireless connections. It measures the network characteristics upon initialization and continuously monitors environmental changes. Network throughput can be estimated by measuring the time duration for sending a certain amount of data as in [4]. Due to the mobile nature of the setup the status of a wireless connection could frequently change (e.g. user moves to other location). Fresh information about a wireless connection is critical for the optimizer to take correct offloading decisions.

The profiler tracks several parameters for the WiFi and 3G interfaces, including the number of packets transmitted and received per second, and receiving and sending data rate [22]. These measurements enable better estimation of the current network performance. We can use the tool Speedtest[2] to measure the mobile network bandwidth.

4.2.3 Energy Profiler

There are two ways to estimate energy consumption, namely, software and hardware monitors. For example, MAUI [3] uses a power meter attached to the battery of a smartphone to build an energy profile. Power Monitor (e.g. Monsoon monitor) is a device that measures energy consumption when data is transmitted from the mobile device to the cloud/edge server based on its power supply to the mobile device.

There are many powerful software-based tools to measure the energy consumption of mobile applications. For example, PowerBooter [49], PowerTutor [50], AppScope [51] or Trepn Profiler [52], which are application frameworks that provide real-time power consumption estimates for power-intensive hardware components including CPU and LCD display as well as GPS, audio, WiFi and cellular interfaces [53]. Although these frameworks do not give as accurate results as a hardware power monitor, their results are still reasonable and do provide useful values because they provide detailed energy consumption information for each hardware component.

4.3 Application Partitioning Processes

Under the the above mentioned challenges of dynamic application, network and energy requirements the workflow of our environment-adaptive application partitioning processes is shown in Fig. 3.

The workflow starts with profiling an application that can be split into multiple tasks. Static analysis and dynamic profiling are applied to the software [30]. We then construct a WCG of the mobile application as shown in Fig. 2(b). Based on cost models, an elastic partitioning algorithm is proposed to make a proper application partitioning.

By using an elastic partitioning algorithm we obtain preliminary partitioning results for response time or energy optimization. During the execution process of the application, if the mobile environment changes, and these changes meet or exceed a certain threshold, the application graph will be re-partitioned according to the new parameters.

2. A free connection analysis tool, which shows real-time download and upload graphs, stores results both locally and on the Internet for sharing, http://www.speedtest.net/

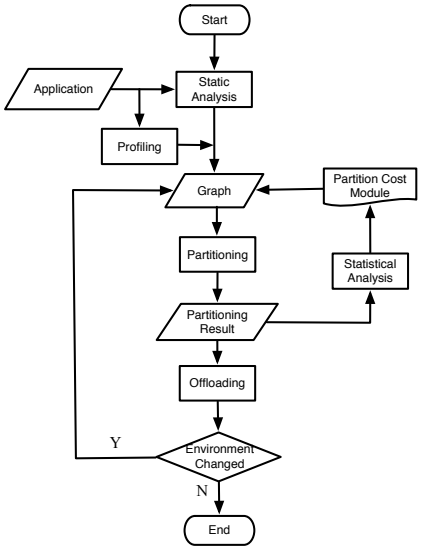


Fig. 3: Flowchart of an application partitioning process

Therefore, we can ultimately achieve the condition-aware and environment-adaptive elastic partitioning. In the context of a mobile environment the decision process includes mobile computing resources inside the device, a battery level, CPU, memory, etc., but also includes an external mobile environment, such as the network connection and the server speed. After partitioning our workflow then automatically offloads the distributed application tasks that require remote execution to a cloud/edge server and executes the remaining tasks locally on the mobile device according to the partitioning results.

5 The Proposed Partitioning Algorithm for Offloading

In this section, we introduce the min-cost offloading partitioning (MCOP) algorithm for WCGs of arbitrary topology. The MCOP algorithm is executed on the mobile device. It takes a WCG as input which represents the operations/-calculations of an application as the nodes and the communication between (e.g. function calls) them as the edges. Each node has two costs: first, the cost of performing the operation locally (e.g., on the mobile device) and, second, the cost of performing it elsewhere (e.g., on the edge/cloud server). The weight of the edges is the communication cost to offload the computation. It is assumed that the communication cost between operations in the same location is negligible. The result contains information about the costs and reports which operations should be performed locally and which should be offloaded.

5.1 Algorithmic Process

The MCOP algorithmic process can be divided into the following two steps:

1) *Unoffloadable Vertices Merging*: An unoffloadable vertex is the one that has special features making it unable to be migrated outside of the mobile device and thus it is located only in the unoffloadable partition. In addition to this, we can choose any task

to be executed locally according to our preferences or other reasons. Then all vertices that are not going to be migrated to the server are merged into one vertex that is selected as the source vertex. By 'merging', we mean that these nodes are coalesced into one, whose weight is the sum of the weights of all merged nodes. Let G represent the original graph after all the unoffloadable vertices are merged.

2) *Coarse Partitioning*: The target of this step is to coarsen G to the coarsest graph $G_{|V|}$. To coarsen means to merge two nodes and reduce the node count by one. Therefore, the algorithm has $|V|-1$ phases. In each phase i (for $1 \leq i \leq |V|-1$), the cut value, i.e. the partitioning cost in a graph $G_i = (V_i, E_i)$ is calculated. G_{i+1} arises from G_i by merging "suitable nodes", where $G_1 = G$. The partitioning result is the minimum cut among all the cuts in an individual phase i and the corresponding task lists for local and remote execution. Furthermore, in each phase i of the coarse partitioning five steps are performed:
   a) Start with A={a}, where a is usually an unoffloadable node in G_i.
   b) Iteratively add the vertex to A that is the most tightly connected to A.
   c) Let s, t be the last two vertices (in order) added to A.
   d) The graph cut of phase i is $(V_i \backslash \{t\}, \{t\})$.
   e) G_{i+1} arises from G_i by merging vertices s and t.

5.2 Merging Function

The *merging* function is used to merge two vertices into one new vertex, which is implemented as in Algorithm 1. If nodes $s, t \in V$ ($s \neq t$), then node s and node t can be merged as follows:

1) Nodes s and t are replaced by a new node $x_{s,t}$. All edges that were previously incident to s or t are now incident to $x_{s,t}$ (except the edge between nodes s and t when they are connected).
2) Multiple edges are resolved by adding edge weights. The weights of the node $x_{s,t}$ are resolved by adding the weights of s and t.

For example, we can merge nodes 2 and 4 in Fig. 4.

5.3 MinCutPhase Function

The *MinCutPhase* function is illustrated in Algorithm 2. The contribution of this algorithm is to make it easy to select the next vertex to be added to the set A, that is *Most Tightly Connected Vertex (MTCV)*, which is defined as the vertex whose $\Delta(v)$ into A is maximal, where $\Delta(v) = w(e(A,v)) - [w^{\text{mobile}}(v) - w^{\text{server}}(v)]$. The total cost from partitioning is:

$$C_{cut(A-t,t)} = C^{\text{mobile}} - \left[w^{\text{mobile}}(t) - w^{\text{server}}(t)\right] + \sum_{v \in A \backslash t} w(e(t,v)), \tag{10}$$

Algorithm 1 The *Merging* function

//This function takes s and t as vertices in the given graph and merges them into one
Function: G'=$Merge(G, w, s, t)$
Input: G: the given graph, $G = (V, E)$
w: the weights of edges and vertices
s, t: two vertices in previous graph that are to be merged
Output: G': the new graph after merging two vertices

```
x_{s,t} ⇐ s ∪ t
for all nodes v ∈ V do
    if v ≠ {s,t} then
        w(e(x_{s,t}, v)) = w(e(s,v)) + w(e(t,v))
        //adding weights of edges
        [w^mobile(x_{s,t}), w^server(x_{s,t})] = [w^mobile(s) + w^mobile(t), w^server(s) + w^server(t)]
        //adding weights of nodes
        E ⇐ E ∪ e(x_{s,t}, v)    //adding edges
    end if
    E' ⇐ E\{e(s,v), e(t,v)}    //deleting edges
end for
V' ⇐ V\{s,t} ∪ x_{s,t}
return G' = (V', E')
```

where $C^{\text{mobile}} = \sum_{v \in V} w^{\text{mobile}}(v)$ is the sum of all local costs and the cut value $C_{cut(A-t,t)}$ is the partitioning cost, $w^{\text{mobile}}(t) - w^{\text{server}}(t)$ is the gain of node t from offloading, and $\sum_{v \in A \backslash t} w(e(t,v))$ is the sum of all extra communication costs due to offloading.

Theorem 1. *$cut(A-t,t)$ is always a minimum $s-t$ cut in the current graph, where s and t are the last two vertices added in this phase, the $s-t$ cut separates nodes s and t to two different sides.*

The run of each *MinCutPhase* function orders the vertices of the current graph linearly, starting with a and ending with s and t, according to the order of addition into A. We want to show that $C_{cut(A-t,t)} \leq C_{cut(H)}$ for any arbitrary $s-t$ cut H.

Lemma 1. *Define H as an arbitrary $s-t$ cut, A_v as a set of vertices added to A before v, and H_v as a cut of $A_v \cup \{v\}$ induced by H. For all active vertices v, we have $C_{cut}(A_v, v) \leq C_{cut}(H_v)$.*

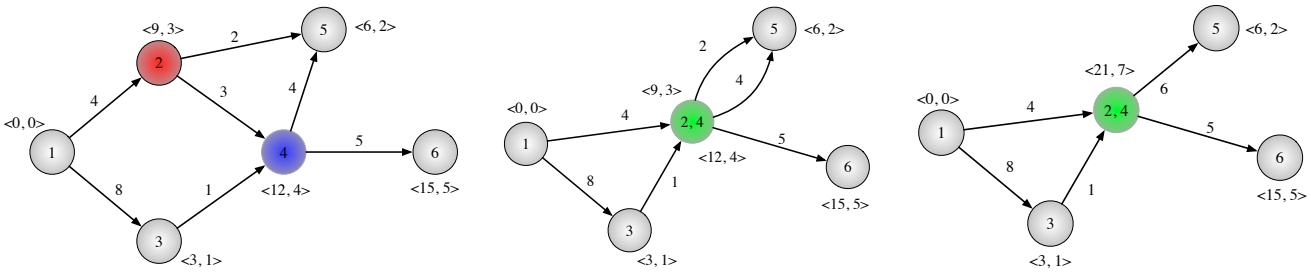
Proof. As shown in Fig. 5, we use induction on the number of active vertices, k.

1) For $k = 1$ the conjecture evaluates to true,
2) Assume the inequality holds true up to u, that is $C_{cut}(A_u, u) \leq C_{cut}(H_u)$,
3) Suppose v is the first active vertex after u, according to the assumption $C_{cut}(A_u, u) \leq C_{cut}(H_u)$, then we have:

$$\begin{aligned} C_{cut}(A_v, v) &= C_{cut}(A_u, v) + C_{cut}(A_v - A_u, v) \\ &\leq C_{cut}(A_u, u) + C_{cut}(A_v - A_u, v) \quad (u \text{ is } MTCV) \\ &\leq C_{cut}(H_u) + C_{cut}(A_v - A_u, v) \\ &\leq C_{cut}(H_v). \end{aligned}$$

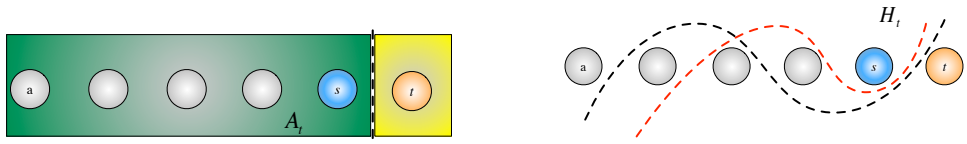
□

Since t is always an active vertex with respect to H, by Lemma 1 we can conclude that $C_{cut(A-t,t)} \leq C_{cut(H)}$ which



(a) Step 1 (b) Step 2 (c) Step 3

Fig. 4: An example of merging two nodes



(a) The $s-t$ cut (b) An arbitrary $s-t$ cut

Fig. 5: Illustration of the proof of Lemma 1

Algorithm 2 The *MinCutPhase* function

//This function performs each phase of the partitioning algorithm

Function: $[cut(A-t,t), s, t]$=$MinCutPhase(G_i, w)$

Input: G_i: the graph in Phase i, i.e., $G_i = (V_i, E_i)$
w: the weights of edges and vertices
SourceVertices: a list of vertices that are forced to be kept on one side of the cut

Output: s, t: the last two vertices that were added to A
$cut(A-t,t)$: the cut between $\{A-t\}$ and $\{t\}$ in phase i

$a \Leftarrow$ arbitrary vertex of G_i
$A \Leftarrow \{a\}$
while $A \neq V_i$ **do**
  $max = -\infty$
  $v_{\max} = null$
  for $v \in V_i$ **do**
    if $v \notin A$ **then**
      //Performance gain by offloading task v to the server
      $\Delta(v) \Leftarrow w(e(A,v)) - [w^{\text{mobile}}(v) - w^{\text{server}}(v)]$
      //Find the vertex most tightly connected to A
      if $max < \Delta(v)$ **then**
        $max = \Delta(v)$
        $v_{max} = v$
      end if
    end if
  end for
  $A \Leftarrow A \cup \{v_{max}\}$
  $a \Leftarrow Merge(G, w, a, v_{max})$
end while
$t \Leftarrow$ the last vertex (in order) added to A
$s \Leftarrow$ the second last vertex (in order) added to A
return $cut(A-t,t)$

means that the cost of $cut(A-t,t)$ is at most as high as the cost of $cut(H)$. This proves Theorem 1.

5.4 MinCut Function and Computational Complexity

The *MinCut* function is illustrated in Algorithm 3. In each phase i it calls the *MinCutPhase* function as described in Algorithm 2. Since some tasks have to be executed locally we need to merge them into one node.

Algorithm 3 The $MinCut$ function

//This function performs an optimal partitioning algorithm

Function: $[minCut, MinCutGroupsList] = MinCut(G, w, SourceVertices)$

Input: G: the given graph, $G = (V, E)$
w: the weights of edges and vertices
SourceVertices: a list of vertices that are forced to be kept on one side of the cut

Output: *minCut*: the minimum sum of weights of edges and vertices along the cut
MinCutGroupsList: two lists of vertices, one local list and one remote list

$w(minCut) \Leftarrow \infty$
for $i = 1 : length(SourceVertices)$ **do**
  //Merge all the source vertices (unoffloadable) into one
  $(G, w) = Merge(G, w, SourceVertices(1), SourceVertices(i))$
end for
while $|V| > 1$ **do**
  $[cut(A-t,t), s, t] = MinCutPhase(G, w)$
  if $w(cut(A-t,t)) < w(minCut)$ **then**
    $minCut \Leftarrow cut(A-t,t)$
  end if
  $Merge(G, w, s, t)$
  //Merge the last two vertices (in order) into one
end while
return $minCut$ and $MinCutGroupsList$

As the runtime of the algorithm *MinCut* is essentially equal to the accumulated runtime of the $|V|-1$ runs of the *MinCutPhase*, which is called on graphs with decreasing number of vertices and edges, it suffices to show that a single *MinCutPhase* needs at most $O(|V|\log|V| + |E|)$ time. The computational complexity of the MCOP algorithm can be formulated as $O(|V|^2 \log|V| + |V||E|)$.

As a comparison, linear programming (LP) solvers are widely used in schemes like [3] and [4]. The LP solver is based on branch and bound, which is an algorithm design paradigm for discrete and combinatorial optimization

problems, as well as general real valued problems [54]. The number of its optional solutions grows exponentially with the number of tasks, which means it has higher time complexity $O\left(2^{|V|}\right)$. Some partitioning solutions such as MAUI [3] have exponential time complexity because they use LP which is not efficient when the number of tasks within the application is large.

In contrast, the MCOP algorithm only has low-order polynomial runtime in the number of tasks. It is very efficient on larger call graphs which demonstrates its advantage over simple partitioning models as used in MAUI: it can group tasks that process large amounts of data on one side, either the server or the mobile side, depending on the network condition.

5.5 Case Study and Discussion

Figure 6 shows that node a is defined as the starting point in which the corresponding task will always be computed by the mobile device. We have s = d and t = f, and the induced ordering a, c, b, e, d, f of the vertices. Node f is cut off from the graph. The first *cut-of-the-phase* corresponds to the partitions {a, c, b, e, d} and {f}. Since the overall local cost is $C^{\text{mobile}} = \sum_{v \in V} w^{\text{mobile}}(v) = 45$, we can calculate the cut cost by using (10) as: $C_{cut(A-f,f)} = 45 - (15 - 5) + 5 = 40$. At the end, we merge nodes s = d and t = f into one.

From Figs. 7-10, we repeat the same process of the *MinCutPhase* function as we did for the first phase in Fig. 6. There are $|V| - 1 = 5$ phases, and at the end, all nodes are merged into one. Then, we compare all the cost values for the different cuts. The minimum value refers to the phase which has the optimal partitioning cut. In this scenario, the minimum cut of graph G is the fourth *cut-of-the-phase*. The optimal cut is between {a, c} and {b, d, e, f} as depicted in Fig. 11 with the minimum cost of $C_{cut(\{\text{a, c}\},\{\text{b, d, e, f}\})} = 45 - (42 - 14) + (4 + 1) = 22$. Here, tasks b, d, e, f are offloaded to the cloud/edge server while tasks a and c are executed locally.

When the execution sequence of tasks of a mobile application is not linear, i.e. it contains tasks that can execute in parallel, the proposed MCOP algorithm can take advantage of this parallelism. It can further reduce the response time of the application or save energy in the mobile device, by partitioning such tasks to different devices or to different processing cores of a single device so that they can be executed in parallel.

6 Performance Evaluation

The performance evaluation results encompass comparisons with other existing schemes, in contrast to the energy conservation efficiency and execution time.

6.1 Setup

Static analysis and dynamic profiling can be combined to construct the WCG of an application. We take a face recognition application[1] as an example. By analyzing this application with Soot, the call graph can be constructed as the tree-based topology shown in Fig. 12. We can obtain the remote estimated execution time by dividing the local estimated execution time by the speedup factor F. When offloading a task to the server the communication cost incurred between the mobile device and the server is computed as the data transfer divided by the bandwidth. Finally, with remote execution and transmission costs, we now have all information to determine the WCG.

To evaluate the partitioning algorithm, we need to know three different kinds of values:

- *Fixed Values*: set by the mobile application developer, determined based on a large number of experiments. For example, the power consumption values of P_m, P_i, and P_{tr} are parameters specific to the mobile system. We use the following values [13]: $P_m \approx 0.9$ W, $P_i \approx 0.3$ W, and $P_{tr} \approx 1.3$ W.
- *Specific Values*: such parameters represent some state of a mobile device, e.g., the size of transferred data, the current wireless bandwidth B (for convenience, we assume $B_{\text{upload}} = B_{\text{download}}$) and the speedup factor F which depends on the compute speed of the current server and the mobile device.
- *Calculated Values*: these values cannot be determined by application developers. For a given application, the computation cost is affected by input parameters and device characteristics, which can be measured using a program profiler. The communication cost is related to transmitting codes/data via wireless interfaces such as WiFi or 3G, which can be tracked by a network profiler.

We compare the partitioning results with two other intuitive strategies without partitioning and, for ease of reference, list all three kinds of offloading techniques:

- *Non Offloading* (*Local Execution*): all computation tasks of an application run locally on the mobile device and there is no communication cost. This may be costly since the mobile device is limited in processing speed and battery life as compared to the powerful computing capability at the server side.
- *Full Offloading* (*Remote Execution*): all computation tasks of mobile applications (except the unoffloadable tasks) are moved from the mobile device to the cloud/edge server for execution [28]. This may significantly reduce the implementation complexity, which makes the mobile devices lighter and smaller. However, full offloading is not always the optimal choice since different application tasks may have different characteristics that make them more or less suitable for offloading [23].
- *Partial Offloading* (*Flexible Execution*): with help of the MCOP algorithm all tasks including unoffloadable and offloadable ones are partitioned into two sets, one for local execution on the mobile device and the other for remote execution on a server node. Before a task is executed, it may require a certain amount of data from other tasks. Thus, data migration via wireless networks is needed between tasks that are executed at different sides.

1. The face recognition application is built using an open source code http://darnok.org/programming/face-recognition/, which implements the Eigenface face recognition algorithm

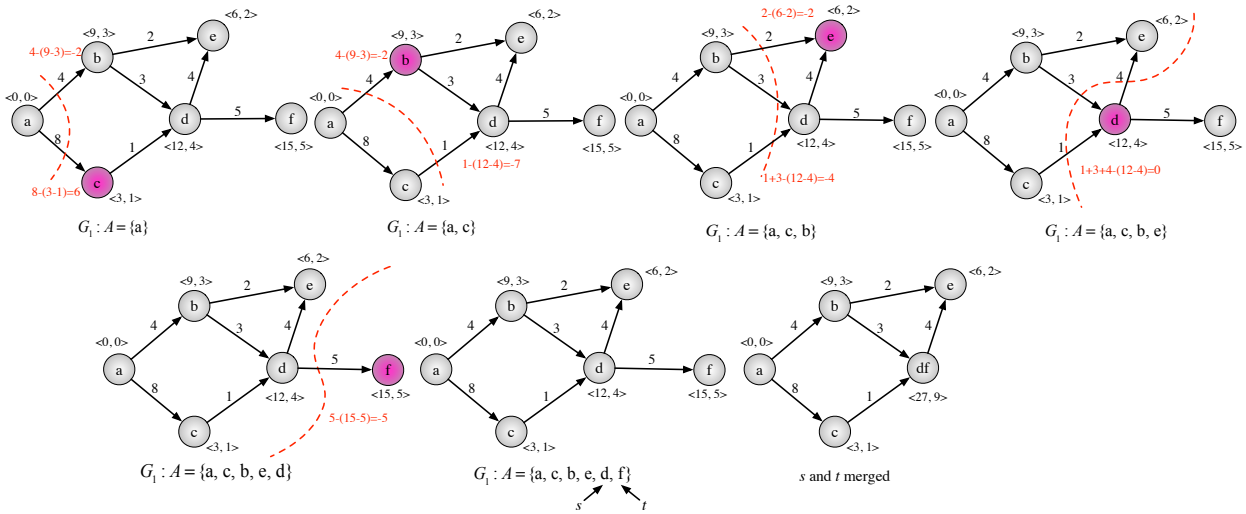


Fig. 6: The 1st phase of *MinCutPhase* function. The induced ordering of the vertices is a, c, b, e, s, t, where s = d and t = f. The 1st cut-of-the-phase corresponds to the partitions {a, c, b, e, d} and {f} with the cut value: $C_{cut(A-f,f)} = 45 - (15 - 5) + 5 = 40$.

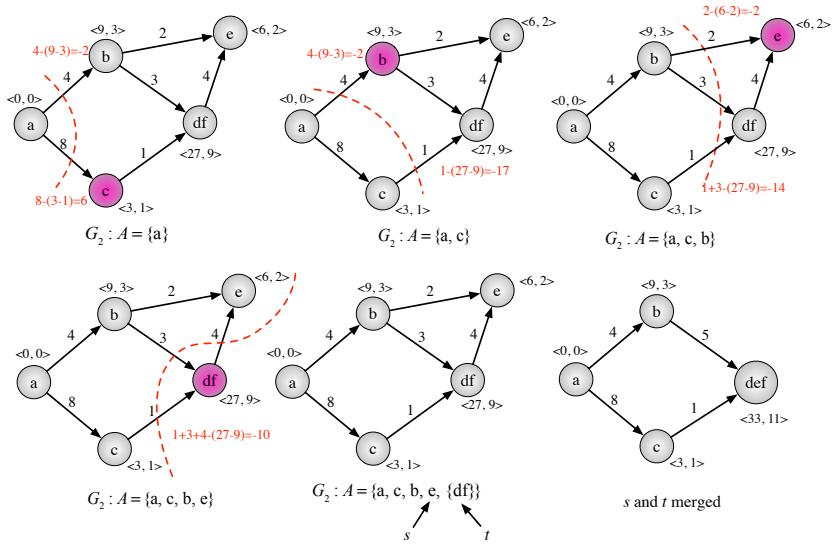


Fig. 7: The 2nd phase of *MinCutPhase* function. The induced ordering of the vertices is a, c, b, s, t, where s = e and t = {df}. The 2nd cut-of-the-phase corresponds to the partitions {a, c, b, e} and {d, f} with the cut value: $C_{cut(A-\{d,f\},\{d,f\})} = 45 - (27 - 9) + (1 + 3 + 4) = 35$.

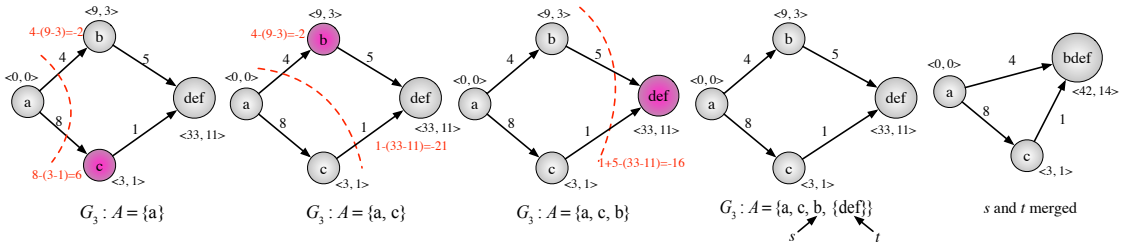


Fig. 8: The 3rd phase of *MinCutPhase* function. The induced ordering of the vertices is a, c, s, t, where s = b and t = {def}. The 3rd cut-of-the-phase corresponds to the partitions {a, b, c} and {d, e, f} with the cut value: $C_{cut(\{a,b,c\},\{d,e,f\})} = 45-(33-11)+(1+5) = 29$.

We define the saved cost in the partial offloading scheme compared to that in the no offloading scheme as *Offloading Gain*, which can be formulated as:

$$\textit{Offloading Gain} = 1 - \frac{\textit{Partial Offloading Cost}}{\textit{Non Offloading Cost}} \cdot 100\%. \quad (11)$$

The offloading gain in terms of time, energy and the weighted sum of time and energy is defined as (5), (7) and (9), respectively.

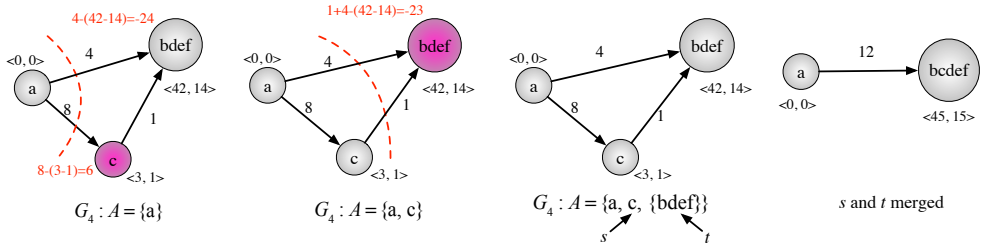


Fig. 9: The 4th phase of *MinCutPhase* function. The induced ordering of the vertices is a, s, t, where s = c and t = {bdef}. The 4th cut-of-the-phase corresponds to the partitions {a, c} and {b, d, e, f} with the cut value: $C_{cut(\{a,c\},\{b,d,e,f\})} = 45 - \{(42-14)-(1+4)\} = 22$.

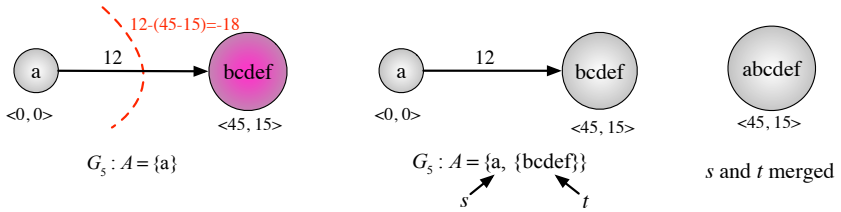


Fig. 10: The 5th phase of *MinCutPhase* function. The induced ordering of the vertices is s, t, where s = a and t = {bcdef}. The 5th cut-of-the-phase corresponds to the partitions {a}, and {b, c, d, e, f} with cut value $C_{cut(\{a\},\{b,c,d,e,f\})} = 45 - (45 - 15) + 12 = 27$.

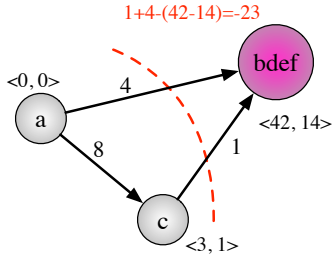


Fig. 11: The optimal cut in phase 4

6.2 Computational Complexity

We have implemented the MCOP algorithm in Java[4] that can serve to illustrate the theoretical results. As an example, we partition the constructed weighted consumption graph on the basis of Fig. 12 under the condition of the speedup factor $F = 20$ and the bandwidth $B = 10$ MB/s, where the *main* and *submit* methods are assumed as unoffloadable nodes, since the task of the *main* method is the trigger of the application that should be unconditionally executed locally on the mobile device and the *submit* method is data-intensive task that requires large data communication between the mobile device and the server. Thus they can be both treated as unoffloadable nodes. The optimal partitioning result is depicted in Fig. 13. The red nodes represent the application tasks that should be offloaded to the cloud/edge server and the blue nodes are the tasks that are supposed to be executed locally on the mobile device. The partitioning results will change as B or F vary.

We compare partial offloading exploiting the proposed MCOP algorithm with other partial offloading schemes. The runtime of the java implementation under a different number of application tasks is depicted as Fig. 14. The overall complexity of the partitioning algorithm proposed in [55] is $O(|V|^3)$ under the one-climb policy (i.e., the execution only migrates once between the mobile device and the cloud clone if ever) and $O(|V|^2 \log^2 |V|)$ under the LARAC algorithm [28]. Further, they are compared with the theoretic computational complexity denoted as $O(|V|^2 \log |V| + |V||E|)$ in Section 5.4.

From Fig. 14, we see that the MCOP algorithm has much lower time complexity when compared to the existing algorithms in [55] and [28]. It can achieve an optimal offloading strategy in minimal time. We also find that both the simulation and the theoretical results match well, which further illustrates that our partitioning algorithm has much lower time complexity than the LP solver which has exponential time complexity.

6.3 Evaluation in Dynamic Conditions

We have built a graphical user interface (GUI) in MATLAB as shown in Fig. 15. The GUI is responsible for user interaction such as receiving input parameters and displaying the application partitioning results.

The user first inputs or selects the relative parameters, such as the *Application Graph*, the *Unoffloadable Nodes* and the *Optimization Model*. We can either use the predefined application graphs of "linear", "loop", "tree" and "mesh" or just choose "user" to input any arbitrary consumption graph. Then, by clicking the "Graph" button, a weighted consumption graph will be constructed based on the above parameters. Further, by clicking the "Start Partitioning" button the partitioning process will begin by calling the partitioning algorithm of MCOP. We can obtain the partitioning results such as the *Partial Offloading Cost*, the *Non offloading Cost*, the *Full Offloading Cost* and the *Offloading Gain*. For example, the optimal partitioning graph will appear as shown in Fig. 16 for the case study in Section 5.5.

We perform a simple simulation using the WCG as depicted in Fig. 4. We have received different results under the different values of the speedup factor F and reliable wireless bandwidth B. The partitioning results will change as B or F vary.

4. An optimal partitioning algorithm, the code can be found in https://github.com/carlosmn/work-offload

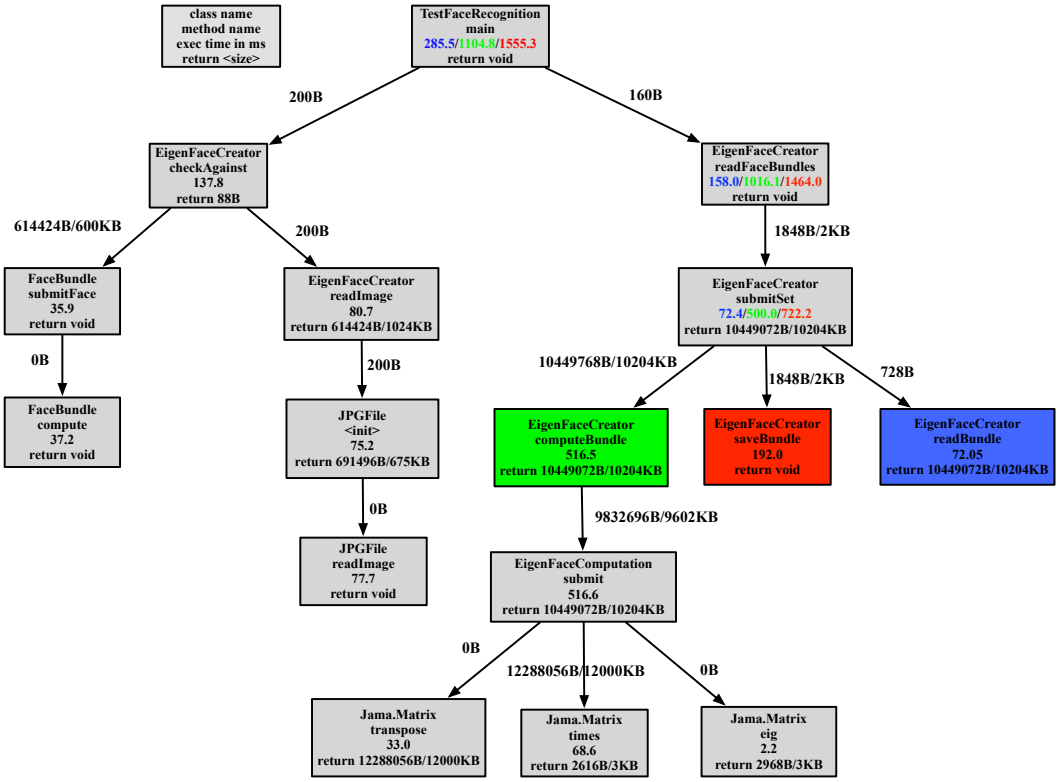


Fig. 12: A call graph for the face recognition application constructed with Soot

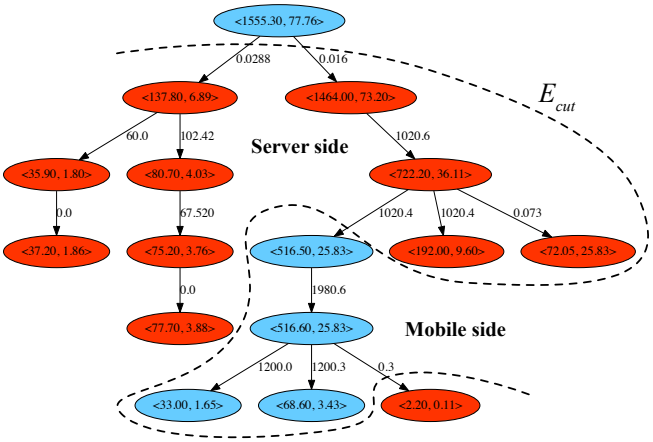


Fig. 13: The optimal partitioning result in the WCG when the speedup factor $F = 20$ and the bandwidth $B = 10$ MB/s

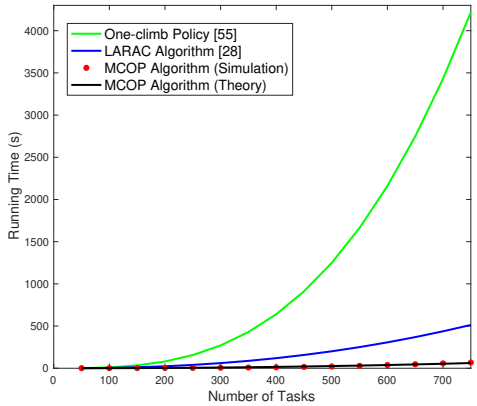


Fig. 14: Comparison of running time for different partitioning algorithms with the number of tasks

In Fig. 17 the speedup factor is set to $F = 3$. Since the low bandwidth results in much higher cost for data transmission, the full offloading scheme cannot benefit from offloading. Given a relatively high bandwidth and stable network, the response time or energy consumption obtained by the full offloading scheme slowly approaches the partial offloading scheme because the optimal partition includes more and more tasks running on the server side until all offloadable tasks are offloaded to the server. With higher bandwidth and a more stable network, they begin to coincide with each other and only decrease because all possible nodes are offloaded and the transmission becomes faster. Both, response time and energy consumption have the same trend as the wireless bandwidth increases. Therefore, high bandwidth and network reliability are crucial for offloading. The mobile system can greatly benefit from offloading in a stable, high bandwidth environment, while with low bandwidth and a fragile network, the *non offloading* performs best.

In Fig. 18 the bandwidth is fixed to $B = 3$ MB/s. It can be seen that offloading benefits from higher speedup factors. When F is very small, the full offloading scheme can reduce energy consumption of the mobile device, however the response time is much higher than in the non offloading scheme. The partial offloading scheme that adopts the MCOP algorithm can effectively reduce execution time and energy consumption, while adapting to environmental changes.

From Figs. 17-18, we can see that the full offloading scheme performs much better than the *non offloading*

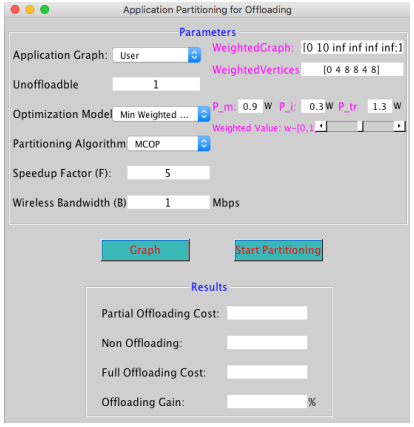


Fig. 15: The user interface for demonstration

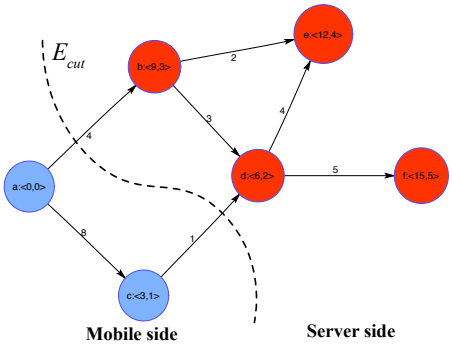


Fig. 16: An optimal partitioning result with the MCOP algorithm

scheme under certain adequate wireless network conditions, because the execution cost of running methods on the cloud/edge server is significantly lower than on the mobile device when the speedup factor F is high. The partial offloading scheme outperforms the *non offloading* and *full offloading* schemes and significantly improves the application performance, since it effectively avoids offloading tasks in the case of large communication cost between consecutive tasks compared to the full offloading scheme, and offloads more appropriate tasks to the server. In other words, neither running all tasks locally on the mobile terminal nor always offloading their execution to a remote server can offer an efficient solution, but our partial offloading scheme can do and thus is much more flexible.

In Fig. 19(a) when the bandwidth is low, the offloading gain for all three cost models is very small and almost identical. That is because more time/energy will be spent in transferring the same amount of data due to the poor network and low bandwidth, resulting in increased execution cost. As the bandwidth increases, the offloading gain first rises drastically and then the increase becomes slower. It can be concluded that the optimal partitioning plan includes more and more tasks running on the server side until all the tasks are offloaded to the server when the network condition and bandwidth increases. In Fig. 19(b) when F is small, i.e. the mobile device and the remote server are almost equally fast, the offloading gain for all three cost models is very low. As F increases, the offloading gain first increases drastically and then saturates. An investment in a server that is more than eight times as fast as the mobile device does not pay off in the offloading experiment because the communication overhead dominates the benefit from offloading. We see from Fig. 19 that the proposed MCOP algorithm can reduce the energy consumption as well as the execution time of an application. Further, it can adapt to environmental changes to some extent and avoid a sharp decline in application performance once the network deteriorates and the bandwidth decreases.

7 Conclusion

To tackle the problem of dynamic application partitioning in a mobile environment, we have proposed a novel offloading partitioning algorithm (MCOP algorithm) that finds the optimal application partitioning under different cost models and arrives at the best tradeoffs between saving time/energy and minimizing transmission costs/delay. This is achieved by constructing a weighted call graphs (WCG) of different topology for software applications under different scenarios. In contrast to the traditional graph partitioning problem our algorithm is not restricted to balanced partitions but takes the infrastructure heterogeneity into account.

The MCOP algorithm possesses a stable quadratic runtime complexity to determine which parts of an application should be offloaded to the cloud/edge server and which parts should be executed locally in order to minimize the energy consumption on the mobile device or to reduce the execution time of an application. Since the reliability of wireless bandwidth can vary due to mobility and interference it strongly affects the optimal partitioning result of the application. Experimental results show that according to environmental changes (e.g., network bandwidth and server speed), the proposed algorithm can effectively achieve the optimal partitioning result in terms of time and energy saving. Offloading benefits a lot from high bandwidths and large speedup factors, while low bandwidth favors the *non offloading* scheme.

In future studies, we will offload several tasks in a simultaneous manner or parallelism way by partitioning such tasks to different devices or processing cores of a single device. In addition, we will build a cloud/edge-based mobile application platform and evaluate the performance of our proposed partitioning algorithm for real mobile applications.

References

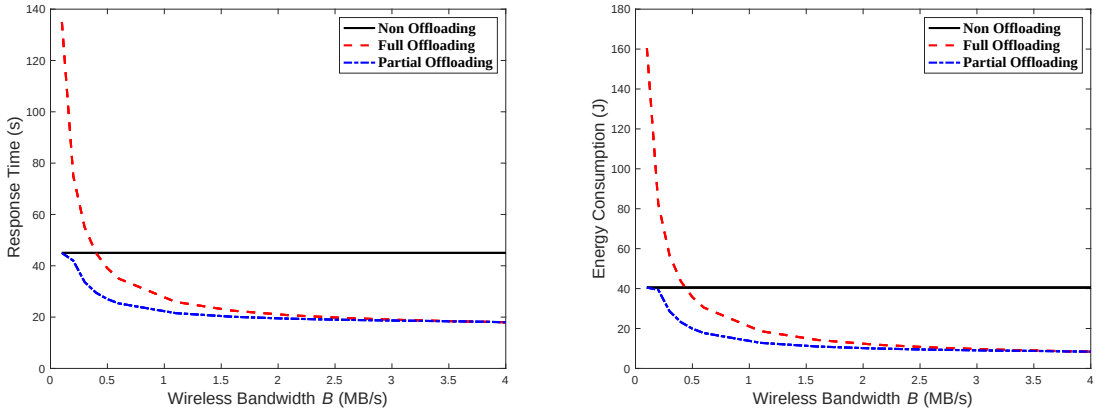
[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[2] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 4, pp. 974–983, 2015.

[3] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62, ACM, 2010.

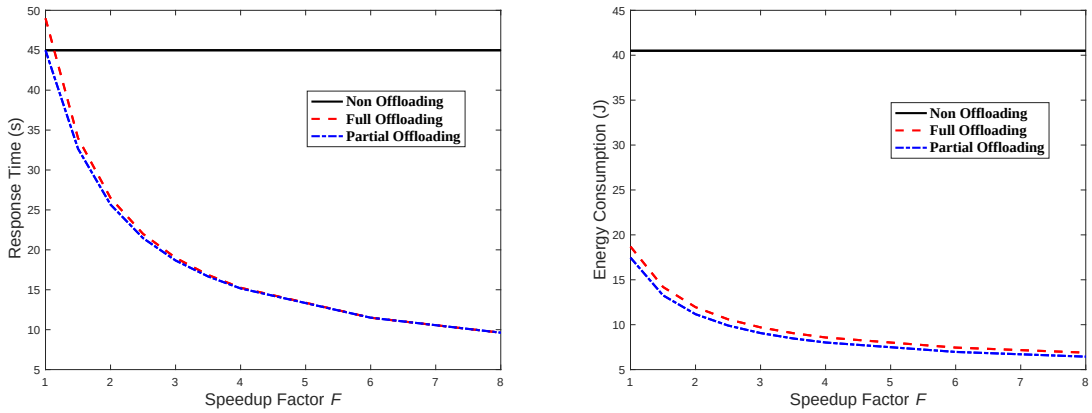
[4] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, pp. 301–314, ACM, 2011.

[5] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel computing*, vol. 26, no. 12, pp. 1519–1534, 2000.



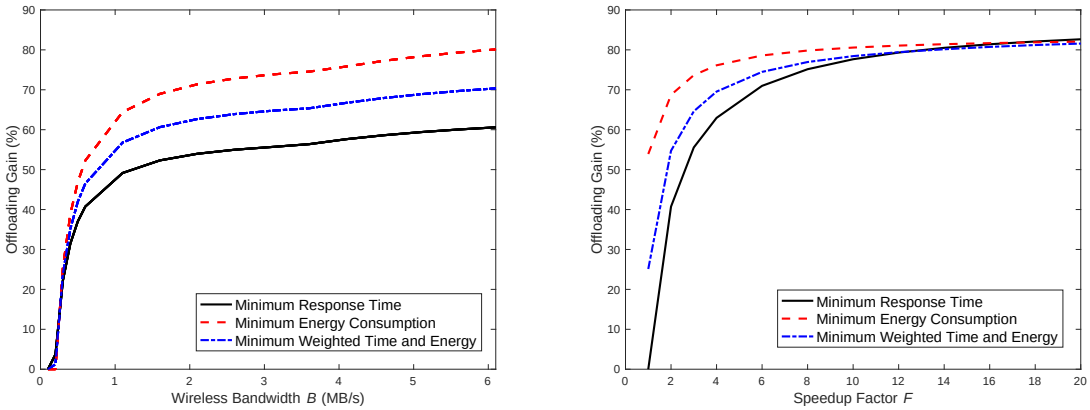
(a) Response Time (b) Energy Consumption

Fig. 17: Comparisons of different schemes under different wireless bandwidth for speedup factor $F = 3$



(a) Response Time (b) Energy Consumption

Fig. 18: Comparisons of different schemes under different speedup factor when the bandwidth $B = 3$ MB/s



(a) Wireless bandwidth B ($F = 3$) (b) Speedup factor F ($B = 3$ MB/s)

Fig. 19: Offloading gains under different environment conditions when $\omega = 0.5$

[6] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM (JACM)*, vol. 44, no. 4, pp. 585–591, 1997.

[7] K. Ali and O. Lhoták, "Application-only call graph construction," in *ECOOP 2012–Object-Oriented Programming*, pp. 688–712, Springer, 2012.

[8] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 11, pp. 1222–1239, 2001.

[9] H. Wu, W. Knottenbelt, K. Wolter, and Y. Sun, "An optimal offloading partitioning algorithm in mobile cloud computing," in *International Conference on Quantitative Evaluation of Systems*, pp. 311–328, Springer, 2016.

[10] H. Wu, Y. Sun, and K. Wolter, "Energy-efficient decision making for mobile cloud offloading," *IEEE Transactions on Cloud Computing*, 2018.

[11] Y. Liu and M. J. Lee, "An effective dynamic programming offload-

ing algorithm in mobile cloud computing system," in *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, pp. 1868–1873, IEEE, 2014.

[12] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[13] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[14] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 23–32, 2013.

[15] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: an efficient code partition algorithm for mobile cloud computing," in *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on*, pp. 80–86, IEEE, 2012.

[16] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: enabling mobile phones as interfaces to cloud applications," in *Middleware 2009*, pp. 83–102, Springer, 2009.

[17] D. Kovachev, "Framework for computation offloading in mobile cloud computing," *IJIMAI*, vol. 1, no. 7, pp. 6–15, 2012.

[18] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading for pervasive computing," *Pervasive Computing, IEEE*, vol. 3, no. 3, pp. 66–73, 2004.

[19] L. Wang and M. Franz, "Automatic partitioning of object-oriented programs for resource-constrained mobile devices with multiple distribution objectives," in *Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on*, pp. 369–376, IEEE, 2008.

[20] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *Wireless Communications, IEEE Transactions on*, vol. 11, no. 6, pp. 1991–1995, 2012.

[21] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: a partition scheme," in *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, pp. 238–246, ACM, 2001.

[22] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM, 2012 Proceedings IEEE*, pp. 945–953, IEEE, 2012.

[23] L. Lei, Z. Zhong, K. Zheng, J. Chen, and H. Meng, "Challenges on wireless heterogeneous networks for mobile cloud computing," *Wireless Communications, IEEE*, vol. 20, no. 3, 2013.

[24] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3317–3329, 2015.

[25] M. P. S. Nir, *Scalable resource augmentation for bobile devices*. PhD thesis, Carleton University Ottawa, 2014.

[26] L. Yang, J. Cao, S. Tang, D. Han, and N. Suri, "Run time application repartitioning in dynamic mobile cloud environments," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp. 336–348, 2016.

[27] V. Haghighi and N. S. Moayedian, "An offloading strategy in mobile cloud computing considering energy and delay constraints," *IEEE Access*, vol. 6, pp. 11849–11861, 2018.

[28] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 708–719, 2018.

[29] E. Abebe and C. Ryan, "Adaptive application offloading using distributed abstract class graphs in mobile environments," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2755–2769, 2012.

[30] J. Niu, W. Song, and M. Atiquzzaman, "Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications," *Journal of Network and Computer Applications*, vol. 37, pp. 334–347, 2014.

[31] T. Verbelen, T. Stevens, F. De Turck, and B. Dhoedt, "Graph partitioning algorithms for optimizing software deployment in mobile cloud computing," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 451–459, 2013.

[32] H. Wu, Q. Wang, and K. Wolter, "Tradeoff between performance improvement and energy saving in mobile cloud offloading systems," in *Communications Workshops (ICC), 2013 IEEE International Conference on*, pp. 728–732, IEEE, 2013.

[33] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[34] H. Wu, Q. Wang, and K. Wolter, "Optimal cloud-path selection in mobile cloud offloading systems based on qos criteria," *International Journal of Grid and High Performance Computing (IJGHPC)*, vol. 5, no. 4, pp. 30–47, 2013.

[35] V. Pandey, S. Singh, and S. Tapaswi, "Energy and time efficient algorithm for cloud offloading using dynamic profiling," *Wireless Personal Communications*, pp. 1–15, 2014.

[36] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pp. 352–357, IEEE, 2014.

[37] A.-C. OLTEANU and N. ŢĂPUŞ, "Tools for empirical and operational analysis of mobile offloading in loop-based applications," *Informatica Economica*, vol. 17, no. 4, pp. 5–17, 2013.

[38] R. Niu, W. Song, and Y. Liu, "An energy-efficient multisite offloading algorithm for mobile devices," *International Journal of Distributed Sensor Networks*, 2013.

[39] K. Sinha and M. Kulkarni, "Techniques for fine-grained, multi-site computation offloading," in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 184–194, IEEE Computer Society, 2011.

[40] B. Y.-H. Kao and B. Krishnamachari, "Optimizing mobile computational offloading with delay constraints," in *Proc. of Global Communication Conference (Globecom 14)*, pp. 8–12, 2014.

[41] M. Goudarzi, M. Zamani, and A. T. Haghighat, "A fast hybrid multi-site computation offloading for mobile cloud computing," *Journal of Network and Computer Applications*, vol. 80, pp. 219–231, 2017.

[42] H. Wu and K. Wolter, "Stochastic analysis of delayed mobile offloading in heterogeneous networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 2, pp. 461–474, 2018.

[43] Y.-W. Kwon and E. Tilevich, "Energy-efficient and fault-tolerant distributed mobile execution," in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pp. 586–595, IEEE, 2012.

[44] S. Ou, K. Yang, and A. Liotta, "An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems," in *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on*, pp. 10–pp, IEEE, 2006.

[45] E. Hyytiä, T. Spyropoulos, and J. Ott, "Offload (only) the right jobs: Robust offloading using the markov decision processes," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*, pp. 1–9, IEEE, 2015.

[46] O. Lhoták and L. Hendren, "Scaling java points-to analysis using spark," in *Compiler Construction*, pp. 153–169, Springer, 2003.

[47] "Soot: A framework for analyzing and transforming java and android applications," in *http://sable.github.io/soot/*.

[48] W. Binder and J. Hulaas, "Using bytecode instruction counting as portable cpu consumption metric," *Electronic Notes in Theoretical Computer Science*, vol. 153, no. 2, pp. 57–77, 2006.

[49] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 105–114, ACM, 2010.

[50] Z. Yang, "Powertutor: a power monitor for android-based mobile platforms," *EECS, University of Michigan, retrieved September*, vol. 2, pp. 19–, 2012.

[51] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring.," in *USENIX Annual Technical Conference*, vol. 12, pp. 1–14, 2012.

[52] Qualcomm, "Trepn profiler." https://developer.qualcomm.com/trepn-profilerAccessed, 6.11 2014.

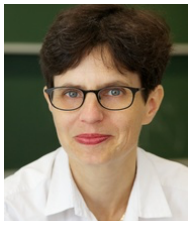
[53] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma, "Modeling, profiling, and debugging the energy consumption of mobile devices," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–40, 2016.

[54] P. M. Narendra and K. Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Transactions on computers*, no. 9, pp. 917–922, 1977.

[55] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 14, no. 1, pp. 81–93, 2015.

Huaming Wu received the B.E. and M.S. degrees from Harbin Institute of Technology, China in 2009 and 2011, respectively, both in electrical engineering. He received the Ph.D. degree with the highest honor in computer science at Freie Universität Berlin, Germany in 2015. He is currently an assistant professor in the Center for Applied Mathematics, Tianjin University. His research interests include model-based evaluation, wireless and mobile network systems, mobile cloud computing and deep learning.

William J. Knottenbelt is a Professor in Applied Performance Modelling in the Department of Computing at Imperial College London, where he became a Lecturer in 2000. His research work focuses on the performance modelling of systems using high-level formalisms, with applications to real-world systems including databases, healthcare systems and data storage infrastructures. His work has been supported by three EPSRC research grants as Principal Investigator and one as Co-Investigator.



Katinka Wolter received her PhD degree from Technische Universität Berlin in 1999. She has been Assistant professor at Humboldt-University Berlin and lecturer at Newcastle University before joining Freie Universität Berlin as a professor for dependable systems in 2012. Her research interests are model-based evaluation and improvement of dependability, security and performance of distributed systems and networks.