An Optimal Offloading Partitioning Algorithm in Mobile Cloud Computing

Huaming Wu[1], William Knottenbelt[2], Katinka Wolter[3], and Yi Sun[3]

[1] Center for Applied Mathematics, Tianjin University, China
whming1986@gmail.com
[2] Department of Computing, Imperial College London, UK
wjk@doc.ic.ac.uk
[3] Institut für Informatik, Freie Universität Berlin, Germany
{katinka.wolter,yi.sun}@fu-berlin.de

Abstract. Application partitioning splits the executions into local and remote parts. Through optimal partitioning, the device can obtain the most benefit from computation offloading. Due to unstable resources at the wireless network (bandwidth fluctuation, network latency, etc.) and at the service nodes (different speed of the mobile device and cloud server, memory, etc.), static partitioning solutions in previous work with fixed bandwidth and speed assumptions are unsuitable for mobile offloading systems. In this paper, we study how to effectively and dynamically partition a given application into local and remote parts, while keeping the total cost as small as possible. We propose a novel min-cost offloading partitioning (MCOP) algorithm that aims at finding the optimal partitioning plan (determine which portions of the application to run on mobile devices and which portions on cloud servers) under different cost models and mobile environments. The simulation results show that the proposed algorithm provides a stable method with low time complexity which can significantly reduce execution time and energy consumption by optimally distributing tasks between mobile devices and cloud servers, and in the meantime, it can well adapt to environmental changes, such as network perturbation.

Keywords: Mobile cloud computing, communication networks, offloading, cost graph, application partitioning

1 Introduction

Along with the maturity of mobile cloud computing, mobile cloud offloading is becoming a promising method to reduce task execution time and prolong battery life of mobile devices. Its main idea is to improve execution by migrating heavy computation from mobile devices to resourceful cloud servers and then receiving the results from them 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.

Offloading all computation components of an application to the remote cloud is not always necessary or effective. Especially for some complex applications (e.g., QR-code recognition [1], online social applications [2], health monitoring using body sensor networks [3]) 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 cloud. Offloading decisions [4, 5] 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 [6] and CloneCloud [7], aiming at offloading 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 mobile devices 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 remote cloud. Program execution can naturally be described as a graph in which vertices represent computation that are labelled with the computation costs and edges reflect the sequence of computation labelled with communication costs [8] where 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 remote cloud servers. Traditional graph partitioning algorithms (e.g., [9–11]) 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 cloud server through links with fluctuating reliability.

The problem of whether or not to offload certain parts of an application to the cloud depends on the following factors: CPU speed of mobile device, speed of the cloud server [12], network bandwidth and reliability, and transmission data size. In this paper, we improve the performance of static partitioning by taking unstable network and cloud conditions into consideration. We explore methods of how to deploy such an offloadable application in a more suitable way by dynamically and automatically determining which parts of the application should be computed on the cloud server and which parts should be left on the mobile device to achieve a particular performance and dependability target (low latency, minimization of energy consumption, low response time, in the presence of unreliable links etc.) [13]. We study how to disintegrate and distribute modules of an application between the mobile side and cloud side, and effectively utilize the cloud resources. We construct a weighted consumption graph (WCG) according to estimated computation and communication costs, and further derive a novel *min-cost offloading partitioning (MCOP)* algorithm designed especially for mobile offloading systems. The MCOP algorithm aims at finding the optimal partitioning plan that minimizes a given objective function (response time, energy consumption or the weighted sum of time and energy) and can be applied to WCGs of arbitrary topology.

The remainder of this paper is organized as follows. Section 2 introduces the partitioning models. An optimal partitioning algorithm for arbitrary topology is proposed and investigated in Section 3. Section 4 gives some evaluation and simulation results. Finally, the paper is summarized in Section 5.

2 Partitioning Models

2.1 Classification of Application Tasks

Applications in a mobile device normally consist of several tasks. Since not all the application tasks are suitable for remote execution, they need to be weighed and distinguished as:

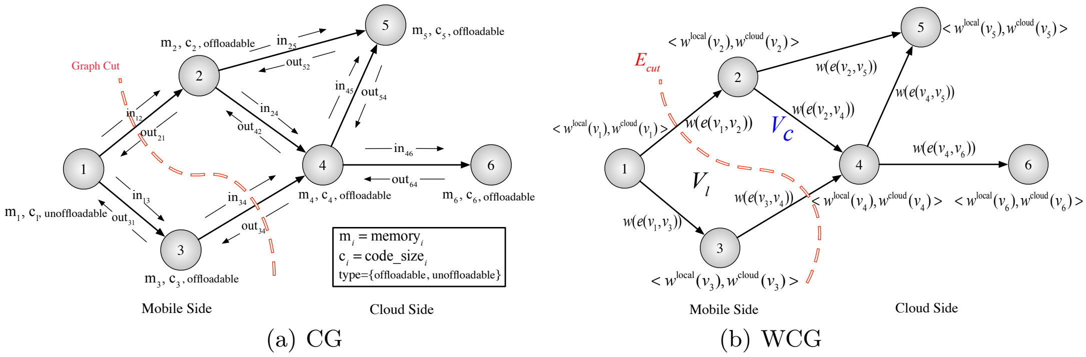
- **Unoffloadable Tasks**: some tasks should be unconditionally executed locally on the mobile device, either because transferring relevant information would use too much time and energy or because these tasks must access local components (camera, GPS, user interface, accelerometer or other sensors etc.) [6]. Tasks that might cause security issues when executed in a different place should not be offloaded either (such as e-commerce). Local processing consumes battery power of the mobile device, but there are no communication costs or delays.
- **Offloadable Tasks**: some application components are flexible tasks that can be processed either locally on the processor of the mobile device, or remotely in a cloud infrastructure. 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 [14].

For unoffloadable components no offloading decisions must be taken. However, as for offloadable ones, since offloading all the application tasks to the remote cloud 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 remote cloud for execution based on available networks, response time or energy consumption. The mobile device has to take an offloading decision based on the result of a dynamic optimization problem.

2.2 Construction of 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 cost on the mobile device and in the cloud in terms of execution time and energy consumption. As cloud servers usually process much faster than mobile devices having a powerful configuration, energy can be saved and performance improved when offloading part of the computation to remote servers [15]. 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 1(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 $e(v_i, v_j) \in$



(a) CG

(b) WCG

Fig. 1. Construction of Consumption Graph (CG) and Weighted Consumption Graph (WCG).

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. 1(b). Each vertex $v \in V$ is annotated with two cost weights: $\boldsymbol{w}(v) =< w^{\text{local}}(v), w^{\text{cloud}}(v) >$, where $w^{\text{local}}(v)$ and $w^{\text{cloud}}(v)$ represent the computation cost of executing the task v locally on the mobile device and remotely on the cloud, respectively. Each vertex is assigned one of the values in the tuple depending on the partitioning result of the resulting application graph [17]. The edge set $E \subset V \times V$ represents the communication cost amongst tasks. The weight of an edge $w(e(v_i, v_j))$ is denoted as:

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

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, and it 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 [18]. Hence, taking the optimal offloading decision is equivalent to partitioning the WCG into those two sets such that an objective function is minimized.

The red dotted line in Fig. 1(b) is one possible partitioning cut, indicating the partitioning of computational workload in the application between the mobile side and the cloud side. V_l and V_c are sets of vertices, where V_l is the local set in which tasks are executed locally at the mobile side and V_c is the cloud set in which tasks are directly offloaded to the cloud. We have $V_l \cap V_c = \emptyset$ and $V_l \cup V_c = V$. Further, E_{cut} is the edge set in which the graph is cut into two parts.

2.3 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 reliability, and the application itself. Device information includes the execution speed of the device and the workloads on it when the application is launched. For a slow device where the aim is to reduce execution time, it is better to offload more computation to the cloud. Network bandwidth and reliability affects data transmission for remote execution. If the bandwidth and reliability 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 cloud.

The partitioning decision is made based on a cost estimate (computation and communication costs) before program execution. On the basis of Fig. 1(b), the overall cost can be calculated as:

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

where C_{total} is the sum of computation costs (local and remote) and communication costs of cut affected edges.

The cloud 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 set of vertices $< V_l, V_c >$ and the cut of edge set E_{cut} in the following way:

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

We seek to find an optimal cut: $\boldsymbol{I}_{\min} = \big\{ I_v, I_e | I_v, I_e \in \{0, 1\} \big\}$ in the WCG such that some application tasks are executed on the mobile side and the remaining ones on the cloud side, while satisifing the general goal of a partition: $\boldsymbol{I}_{\min} = \arg\min_{\boldsymbol{I}} \; C_{\text{total}}(\boldsymbol{I})$. The dynamic execution configuration of an elastic application can be decided based on different saving objectives with respect to response time and energy consumption. Since the communication time and energy cost for the mobile device will vary according to the amount of data to be transmitted and the wireless network conditions. A task's offloading goals may change due to a change in environmental conditions.

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 its cost. If the minimum response time is selected as objective function, we can calculate the total time spent due to offloading as:

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

where $T_v^l = F \cdot T_v^c$ is the computation time of task v on the mobile device when it is executed locally; F is the speedup factor, the ratio of the cloud server's processing speed compared to that of the mobile device. T_v^c is the computation time of task v on the cloud 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.

Minimum Energy Consumption If the minimum energy consumption is chosen as the objective function, we can calculate the total energy consumed due to offloading as:

$$E_{\text{total}}(\boldsymbol{I}) = \underbrace{\sum_{v \in V} I_v \cdot E_v^l}_{\text{local}} + \underbrace{\sum_{v \in V} (1 - I_v) \cdot E_v^i}_{\text{idle}} + \underbrace{\sum_{e \in E} I_e \cdot E_e^{tr}}_{\text{communication}}, \tag{5}$$

where $E_v^l = p_m \cdot T_v^l$ 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 cloud; $E_e = p_{tr} \cdot T_e^{tr}$ is the energy spent on the communication between the mobile device and the cloud including possibly necessary retransmissions; p_m, p_i and p_{tr} are the powers of the mobile device for computing, while being idle and for data transfer, respectively.

Minimum of the Weighted Sum of Time and Energy If we combine both the response time and energy consumption, we can design a cost model for partitioning as follows [19]:

$$W_{\text{total}}(\boldsymbol{I}) = \omega \cdot \frac{T_{\text{total}}(\boldsymbol{I})}{T_{\text{local}}} + (1 - \omega) \cdot \frac{E_{\text{total}}(\boldsymbol{I})}{E_{\text{local}}}, \tag{6}$$

where $0 \leq \omega \leq 1$ is a weighting parameter used to share relative importance between the response time and energy consumption. Large ω favors response time while small ω favors energy consumption [20, 21]. Performance can be traded for power consumption and vice versa [22, 23], therefore we can use ω to express preferences for different applications. If $T_{\text{total}}(\boldsymbol{I})/T_{\text{local}}$ is less than 1, the partitioning will improve the application's performance. Similarly, if $E_{\text{total}}(\boldsymbol{I})/E_{\text{local}}$ is less than 1, it will reduce the energy consumption.

We only perform the partitioning when it is beneficial. Not all applications can benefit from partitioning because of application-specific properties. A pre-calibration of the computation cost on each device is necessary. Offloading is beneficial only if the speedup of the cloud server outweighs the extra communication cost. We compare the partitioning results with two other intuitive strategies without partitioning and, for ease of reference, we list all three kinds of offloading techniques:

- *No Offloading* (*Local Execution*): all computation tasks of an application are running 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 cloud side.

- *Full Offloading*: all computation tasks of mobile applications (except the unoffloadable tasks) are moved from the local mobile device to the remote cloud for execution. 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 [24].
- *Partial Offloading* (*With Partitioning*): with the 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 cloud 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.

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:

$$Offloading\ Gain = 1 - \frac{Partial\ Offloading\ Cost}{No\ Offloading\ Cost} \cdot 100\%. \tag{7}$$

3 Partitioning Algorithm for Offloading

In this section, we introduce the min-cost offloading partitioning (MCOP) algorithm for WCGs of arbitrary topology. The MCOP algorithm takes a WCG as input in which an application's operations/calculations are represented as the nodes and the communication between 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., in the cloud). The weight of the edges is the communication cost to the offloaded computation. We assume that the communication cost between tasks in the same location is negligible. The result contains information about the cost and reports which operations should be performed locally and which should be offloaded.

3.1 Steps

The MCOP algorithm can be divided into two steps as follows:

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. Apart from 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 cloud are merged into one 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 results are the minimum cut among all the cuts in an individual phase i and the corresponding group lists for local and cloud execution. Furthermore, in each phase i of the coarse partitioning we still have five steps:

(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 the phase i is $(V_i \backslash \{t\}, \{t\})$.
(e) G_{i+1} arises from G_i by merging vertices s and t.

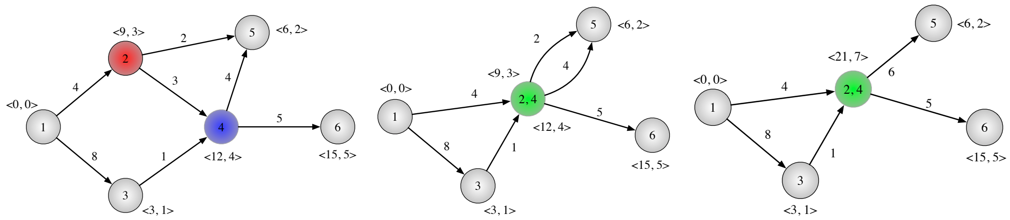
3.2 Algorithmic Process

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

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 they can be merged as follows:

1. Nodes s and t are chosen.
2. 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).
3. 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 as shown in Fig. 2.



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

Fig. 2. An example of merging two nodes

The core of this algorithm is to make it easy to select the next vertex to be added to the local set A. We have the following definition.

Definition 1 *$\exists v \in V \backslash A$, if the potential benefit from offloading once v is offloaded:*

$$\Delta(v) = [w^{local}(v) - w^{cloud}(v)] - w(e(A, v))$$

is the minimum, then task v has the most chance to be executed locally, and the vertex v is called Most Tightly Connected Vertex (MTCV).

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

```
$x_{s,t} \Leftarrow s \cup t$
for all nodes $v \in V$ do
    if $v \neq \{s, t\}$ then
        $w(e(x_{s,t}, v)) = w(e(s, v)) + w(e(t, v))$
        //adding weights of edges
        $\left[w^{\text{local}}(x_{s,t}), w^{\text{cloud}}(x_{s,t})\right] = \left[w^{\text{local}}(s) + w^{\text{local}}(t), w^{\text{cloud}}(s) + w^{\text{cloud}}(t)\right]$
        //adding weights of nodes
        $E \Leftarrow E \cup e(x_{s,t}, v)$    //adding edges
    end if
    $E' \Leftarrow E \backslash \{e(s, v), e(t, v)\}$    //deleting edges
end for
$V' \Leftarrow V \backslash \{s, t\} \cup x_{s,t}$
return $G' = (V', E')$
```

Algorithm 2 The *MinCut* function

//This function performs an optimal offloading partition 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 have to be kept in one side of the cut
Output: *minCut*: the minimum sum of weights of edges and vertices among 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 $MinCutGroupsList$
```

Further, we have the total cost from partitioning:

$$C_{cut(A-t,t)} = C^{\text{local}} - \Big[w^{\text{local}}(t) - w^{\text{cloud}}(t)\Big] + \sum_{v \in A\setminus t} w(e(t,v)), \tag{8}$$

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

Algorithm 3 The *MinCutPhase* function

```
//This function perform one 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 in one side of the cut
Output: $s$, $t$: the lasted two vertices that are added to $A$
        $cut(A-t,t)$: the cut between $\{A-t\}$ and $\{t\}$ in phase $i$

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

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 the phase, the $s-t$ cut separates nodes s and t on 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.

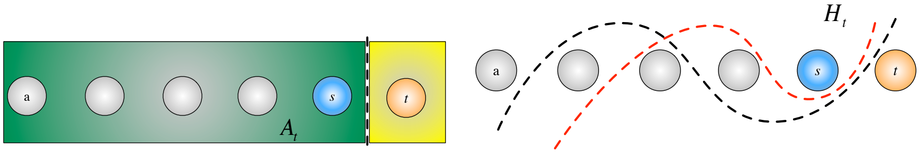
Lemma 1 *We define H as an arbitrary $s-t$ cut, A_v as a set of vertices added to A before v, and H_v as a cut of $A_v \cup \{v\}$ induced by H. For all active vertices v, we have $C_{cut}(A_v, v) \leq C_{cut}(H_v)$.*

Proof. As shown in Fig. 3, we use induction on the number of active vertices, k.

1. When $k = 1$, the claim is 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 that $C_{cut}(A_u, u) \leq C_{cut}(H_u)$, then we have:

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

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



(a) The $s-t$ cut

(b) An arbitrary $s-t$ cut

Fig. 3. The illustration for the proof of Lemma 1

3.3 Computational Complexity

As the running time of the algorithm *MinCut* is essentially equal to the added running time of the $|V|-1$ runs of *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 yielding an overall running time. The computational complexity of the MCOP algorithm can be noted as $O(|V|^2\log|V|+|V||E|)$.

As a comparison, Linear Programming (LP) solvers are widely used in schemes like MAUI [6] and CloneCloud [7]. An 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. The number of its optional solutions grows exponentially with the number of tasks, which means higher time complexity $O\left(2^{|V|}\right)$.

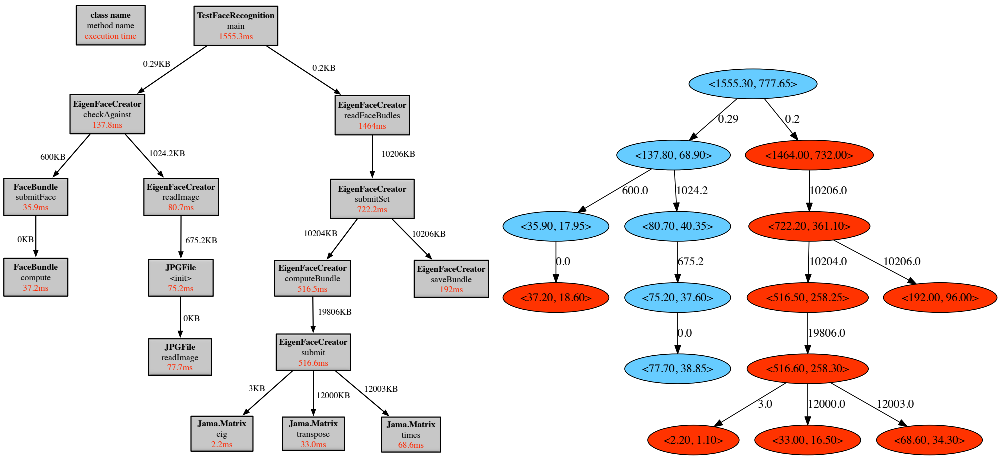
While the partitioning e.g., MAUI has exponential time complexity by using LP, our algorithm only has low-order polynomial run time in the number of tasks. Therefore, the MCOP algorithm can handler larger call graphs, which shows its advantage over simple partitioning models as used in MAUI: it can group tasks that process large amounts of data on one side, either the Cloud or the mobile, depending on the network condition.

4 Performance Evaluation

Comparing the execution time spent on the mobile device and the one on the cloud, the speedup factor F is obtained. In practice, we will first access to the cloud server to estimate the remote execution time. We use the average value, since the mobile device might assign more computation resources to a process at different moments of its execution. Therefore, during runtime of an application the link and node cost is constantly updated (the updated value will be an average of the past values and the newly obtained one).

The construction of WCG closely depends on profiling, i.e., the process of gathering the information required to make 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). Since the focus of this paper is on the partitioning algorithm we will not enter into the details of profiling techniques, which can be found in many existing references [6, 25].

We take a face recognition application[1] as an example. By analyzing this application with Soot [26], the call graph could be built as a tree-based topology shown in Fig. 4(a). We further construct weighted consumption graph under the condition of the speedup factor $F = 2$ and the bandwidth $B = 1$ MB/s with reliability $= 1$, where the *main* and *checkAgainst* methods are assumed as unoffloadable nodes.



(a) A call graph

(b) A partitioning result

Fig. 4. An optimal partitioning result of the face recognition application

We then implement the MCOP algorithm in Java[2]. The optimal partitioning result is depicted in Fig. 4(b), where red nodes represent the application tasks that

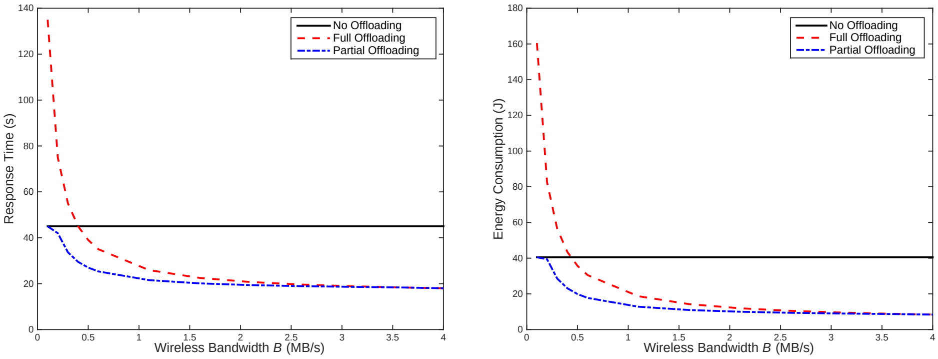
[1] The face recognition application is built using an open source code `http://darnok.org/programming/face-recognition/`, which implements the Eigenface face recognition algorithm

[2] An optimal partitioning algorithm, the code can be found in `https://github.com/carlosmn/work-offload`, thanks to Daniel Seidenstücker and Carlos Martín Nieto for their help

should be offloaded to the remote cloud and the blue nodes are the tasks that are supposed to be executed locally on the mobile device.

We do a simple simulation with the WCG as predicted in Fig. 2. We have received different results under the different parameters of speedup factor F and reliable wireless bandwidth B. The partitioning results will change as B or F vary.

In Fig. 5 the speedup factor is set to $F = 3$. Since the low bandwidth results in much higher cost for data transmission, the full offloading scheme can not benefit from offloading. Given a relatively large 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 cloud side until all offloadable tasks are offloaded to the cloud. With higher bandwidth and more stable network, they begin to coincide with each other and only decrease because all possible nodes are offloaded and the transmissions become faster. Both, response time and energy consumption have the same trend as the wireless bandwidth increases. Therefore, bandwidth and network reliability is a crucial element for offloading since the mobile system could benefit a lot from offloading in stable, high bandwidth environments, while with low bandwidth and fragile network, the *no offloading* scheme is preferred.

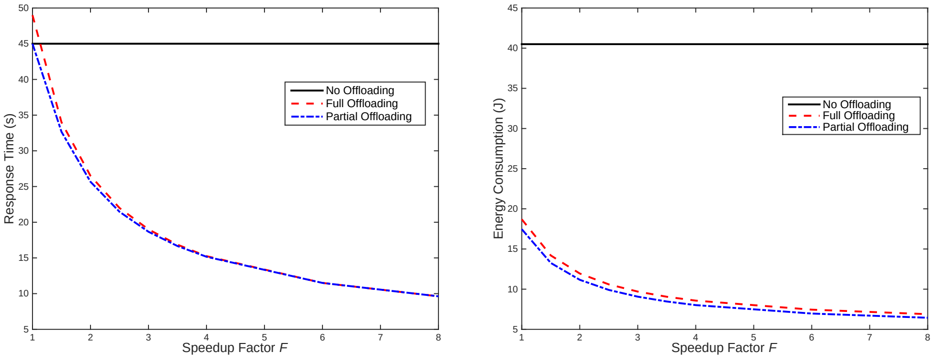


(a) Response Time (b) Energy Consumption

Fig. 5. Comparisons of different schemes under different wireless bandwidths when the speedup factor $F = 3$

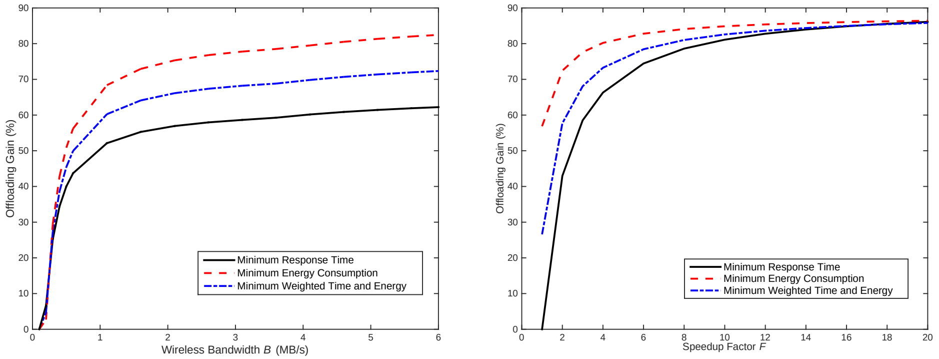
In Fig. 6 the bandwidth is fixed at $B = 3$ MB/s. It can be seen that offloading benefits from higher speedup factors. When F is very small, the full offloading scheme can reduce energy consumption of the mobile device. However, it takes much longer than without offloading. The partial offloading scheme that adopts the MCOP algorithm can effectively reduce execution time and energy consumption, while adapting to environmental changes.

From Figs. 5-6, we can tell that the full offloading scheme performs much better than the *no offloading* scheme under certain adequate wireless network conditions, because the execution cost of running methods on the cloud server is significantly lower than on the mobile device when the speedup factor F is high. The partial offloading scheme outperforms the *no 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



(a) Response Time (b) Energy Consumption

Fig. 6. Comparisons of different schemes under different speedup factors when the bandwidth $B = 3$ MB/s



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

Fig. 7. Offloading gains under different environment conditions when $\omega = 0.5$

the full offloading scheme, and offloads more appropriate tasks to the cloud 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.

In Fig. 7(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 cloud side until all the tasks are offloaded to the cloud when the network condition and bandwidth increases. In Fig. 7(b) when F is small, the offloading gain for all three cost models is very low since a small value means very little computational cost reduction from remote execution. As F increases, the offloading gain first rises drastically and then approaches the same value. That is because the benefits from offloading cannot neglect the extra communication cost. From Fig. 7, the proposed MCOP algorithm is able to effectively reduce the application's energy consumption as well as its execution time. Further, it can adapt to environmental changes to some extent and avoids a sharp decline in application performance once the network deteriorates and the bandwidth decreases.

5 Conclusion

To tackle the problem of dynamic 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 to arrive at the best tradeoff between saving time/energy and transmission costs/delay. Contrary to the traditional graph partitioning problem, our algorithm is not restricted to balanced partitions but takes the infrastructure heterogeneity into account.

The MCOP algorithm provides a stably quadratic runtime complexity for determining which parts of application tasks should be offloaded to the cloud server and which parts should be executed locally, in order to save energy of the mobile device and 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 application's optimal partitioning result. When the network is poor, high communication cost will be incurred, and the MCOP algorithm will include more application tasks for local execution. Experimental results show that according to environmental changes (e.g., network bandwidth and cloud server performance), 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 *no offloading* scheme.

The concept of optimal application partitioning under constraints generalises to many scenarios in distributed computing where should be explored further.

References

1. 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.
2. 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.
3. H. Wu, Q. Wang, and K. Wolter, "Mobile healthcare systems with multi-cloud offloading," in *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*, vol. 2, pp. 188–193, IEEE, 2013.
4. H. Wu, *Analysis of Offloading Decision Making in Mobile Cloud Computing.* PhD thesis, Freie Universität Berlin, 2015.
5. H. Wu, Q. Wang, and K. Wolter, "Methods of cloud-path selection for offloading in mobile cloud computing systems.," in *CloudCom*, pp. 443–448, 2012.
6. 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.
7. 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.

8. B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel computing*, vol. 26, no. 12, pp. 1519–1534, 2000.
9. M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM (JACM)*, vol. 44, no. 4, pp. 585–591, 1997.
10. K. Ali and O. Lhoták, "Application-only call graph construction," in *ECOOP 2012–Object-Oriented Programming*, pp. 688–712, Springer, 2012.
11. 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.
12. Y. Liu and M. J. Lee, "An effective dynamic programming offloading algorithm in mobile cloud computing system," in *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, pp. 1868–1873, IEEE, 2014.
13. H. Wu and K. Wolter, "Software aging in mobile devices: Partial computation offloading as a solution," in *Software Reliability Engineering Workshops (ISSREW), 2015 IEEE International Symposium on*, IEEE, 2015.
14. 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.
15. R. Niu, W. Song, and Y. Liu, "An energy-efficient multisite offloading algorithm for mobile devices," *International Journal of Distributed Sensor Networks*, 2013.
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. 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.
18. 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.
19. H. Wu and K. Wolter, "Tradeoff analysis for mobile cloud offloading based on an additive energy-performance metric," in *Performance Evaluation Methodologies and Tools (VALUETOOLS), 2014 8th International Conference on*, ACM, 2014.
20. H. Wu, Y. Sun, and K. Wolter, "Analysis of the energy-response time tradeoff for delayed mobile cloud offloading," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, pp. 33–35, Sept. 2015.
21. H. Wu, W. Knottenbelt, and K. Wolter, "Analysis of the energy-response time tradeoff for mobile cloud offloading using combined metrics," in *Teletraffic Congress (ITC 27), 2015 27th International*, pp. 134–142, IEEE, 2015.
22. 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.
23. 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.
24. 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.
25. 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.
26. "Soot: A framework for analyzing and transforming java and android applications," in *http://sable.github.io/soot/*.