Energy-Efficient Computation Offloading in Cellular Networks

Yeli Geng*, Wenjie Hu*, Yi Yang*, Wei Gao† and Guohong Cao*
*Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA
†Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN
*{yzg5086, wwh5068, yzy123, gcao}@cse.psu.edu, †weigao@utk.edu

***Abstract*—Computationally intensive applications may quickly drain mobile device batteries. One viable solution to address this problem utilizes computation offloading. The tradeoff is that computation offloading introduces additional communication, with a corresponding energy cost. Yet, previous research into computation offloading has failed to account for the special characteristics of cellular networks that impact mobile device energy consumption. In this paper, we aim to develop energy efficient computation offloading algorithms for cellular networks. We analyze the effects of the long tail problem on task offloading, formalize the computation offloading problem, and use Dijkstra's algorithm to find the optimal decision. Since this optimal solution relies on perfect knowledge of future tasks, we further propose an online algorithm for offloading. We have implemented this latter algorithm on Android-based smartphones. Both experimental results from this implementation and trace-driven simulation show that our algorithm can significantly reduce the energy of computation offloading in cellular networks.**

I. Introduction

Due to the increased processing capability of smartphones in recent years, computationally intensive mobile applications such as image recognition, gaming and speech recognition are becoming increasingly popular. However, these applications also may quickly drain mobile device batteries. One viable solution to address this problem utilizes computation offloading [1]. Offloading computationally intensive tasks to remote resource-rich servers can save a mobile device's energy.

Previous research has investigated how to make offloading decisions, i.e., determining which tasks should be offloaded to minimize the energy consumption of the mobile devices. Some of this work [2], [3], [4] considers the tradeoff between the energy saved by moving computations to the server and the energy consumed to offload the computation. Other researchers propose methods [5], [6] to predict the offloading cost based on the application behavior and execution environment. However, previous work does not account for the special characteristics of cellular networks in making the offloading decision.

In cellular networks, a mobile device's cellular interface transitions between different states, each of which operates at its own power level. The transitions between these states are controlled by several timers, whose timeout values can be as high as several seconds [7]. After having completed a data transmission, the cellular interface will stay in a high-power state for some time before switching to a low-power state. Thus, the cellular interface may consume a substantial amount of energy (referred to as the *long tail* problem) even when there is no network traffic. Recent research [8], [9], [10] has shown that a large fraction of energy may be wasted due to the long tail problem in 3G networks, and that this problem becomes only worse in 4G/LTE [11]. Therefore, the long tail problem should be taken into account when calculating the communication cost of offloading. Although some research has been done to address the long tail problem by deferring and aggregating data traffic [7], [9], [10], [12], none addresses how to select which tasks should be offloaded while accounting for the long tail problem.

In this paper, we aim to develop energy-efficient computation offloading algorithms that are "tail aware", taking into account the unique energy characteristics of cellular networks. Specifically, we attempt to answer the following question: *Given a number of computation tasks, what is the best offloading decision to minimize the energy consumed by the mobile device?* When making the offloading decision, we consider the cost of both communication and computation. However, the long tail problem makes it tricky to calculate the communication cost. Specifically, the energy consumed to offload a task may be affected by the computation offloading of previous tasks. As each task can be executed either locally on the mobile device or remotely on the server through offloading, there are $O(2^n)$ possibilities making a naïve approach to calculating the optimal solution intractable.

To solve this problem, we analyze the effects of the long tail problem on task offloading and show that only a linear number of possible computation offloading decisions need be considered. We then create the decision states for each task which represent all possible offloading decisions for that task. Based on these decision states, we formalize the computation offloading problem, and use Dijkstra's algorithm to find the optimal decision. Since this optimal solution requires perfect knowledge of future tasks (which will not be available in practical scenarios), we further propose an online algorithm for offloading.

The main contributions of this paper are as follows.

- We are the first both to consider the long tail problem when making offloading decisions, and to formalize the problem of energy-efficient computation offloading in cellular networks.
- We propose an optimal offline computation offloading algorithm to minimize the energy consumed by the mobile

device, and design an online version.
- We have implemented the online offloading algorithm on Android-based smartphones. Experimental results show that our offloading algorithm can reduce energy consumption by 35%. Trace-driven simulation shows that the proposed offloading algorithms are more effective for complex tasks with small data size, and that computation offloading is substantially more effective in LTE networks than 3G ones.

The remainder of this paper is organized as follows: In Section II, we briefly survey related work. Section III introduces the problem definition and the system model. Section IV presents our optimal offloading algorithm. The online offloading algorithm is introduced in Section V. Section VI reports experimental results and the performance evaluation of our algorithms. Finally, we conclude in Section VII.

II. Related Work

Offloading computation from mobile devices to the cloud has received considerable attention recently. Some previous work has focused on enabling computation offloading by utilizing recent technology advances on virtualization and mobile cloud computing [13], [14]. For example, MAUI [2] uses the Microsoft .NET to develop a framework that enables fine-grained code offload. CloneCloud [3], [4] creates device clones on the cloud to enable seamless cloud-augmented execution. ThinkAir [15] enforces on-demand VM creation and resource allocation, and performs offloading based on user specified policy.

Beside solving how to offload, much work has been done on what to offload. Such work aims to make appropriate decisions on application partitioning to enable efficient offloading. OLIE [16] profiles network conditions and program behavior at runtime, and then partitions the application and offloads part of the execution to a nearby surrogate. Wolski *et al*. [17] predict statistically when an offloaded computation will perform better than executing it on the mobile device. Odessa [18] adaptively makes offloading decisions for mobile interactive perception applications. However, the goal of OLIE is to overcome the memory constraint, and the other two approaches aim to improve the performance of mobile devices. MAUI [2] proposes a program partitioning strategy that minimizes the smartphone's energy consumption. Zhang *et al*. [19] investigate the scheduling policy of a sequence of tasks, and provide an approximate solution to minimize the energy consumption. Although these studies have similar goals as ours, none of them considers the long tail problem in offloading which is the focus of our work.

Our work is also related to recent advances in the understanding of the power characteristics of cellular networks. Previous work studies the energy characteristics of 3G, and finds that the cellular interface stays in the high power state after completing a data transmission [8], [20]. Later studies find that LTE is even less energy efficient than 3G and the long tail problem is found to be a key contributor [11]. Fast dormancy [21] has been proposed to address the long tail problem. However, fast dormancy requires support from both smartphones and cellular carriers, and it may increase the signaling overhead due to frequent state transitions. Other researchers propose to defer and aggregate the network traffic [7], [9], [10]. Recently, Xiang *et al.* have addressed the long tail problem when multiple applications are offloaded. They propose the technique of coalesced offloading, which coordinates the offloading requests of multiple applications to save energy. However, they do not address the problem of which parts (tasks) of the application should be offloaded. Different from them, our work focuses on how to make optimal offloading decisions to save energy.

III. Problem Definition and System Model

In this section, we introduce the problem definition and the system model of computation offloading.

A. Offloading Scenario

We consider the following scenario: A user runs computationally intensive applications on his mobile device and generates multiple tasks. Some of the tasks will be executed on the mobile device and some of them will be executed on the server to save energy. One example of such application is the Optical Character Recognition (OCR) which can automatically recognize characters in images taken by a smartphone and output readable text. If the task only has a small number of words, it can be finished in a short time on the smartphone, such as mobile apps for finding the road signs. However, if the task has images of several pages of a research paper, the computation on the smartphone will take much longer time and consume lots of energy. To offload the task, the smartphone sends the image to the server, lets the server perform character recognition and finally receives the text. In this way, the execution time and the energy consumption can be reduced.

To simplify our analysis, we first assume sequential task scenarios and then discuss how to adapt the proposed approach for parallel tasks in Section IV.

B. Power Model of the Cellular Interface

Cellular networks employ a Radio Resource Control (RRC) state machine for each mobile device. In 3G networks, the cellular interface has three states: IDLE, DCH and FACH. Fig. 1 illustrates the power level of a smartphone when using the 3G interface to transmit data.

Initially, the radio interface stays in IDLE state when there is no network activity, consuming very low power. When there are packets to be sent or received, the interface transits to DCH state by first establishing a signaling connection with the backbone network, and then obtaining dedicated transmission channels. This process requires control message exchanges between the smartphone and the backbone network, and the incurred time delay is called the *promotion delay* (t_{pro}). The radio interface remains in the DCH state during data transmission. An inactivity timer is triggered when the transmission completes. The radio interface will transit to the FACH state when the timer expires, and finally transit to the

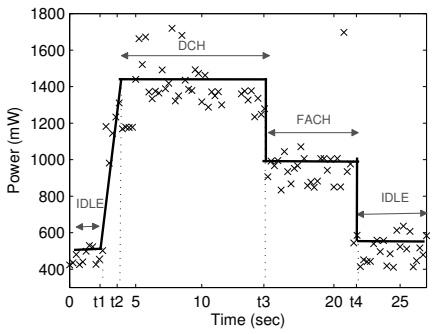


Fig. 1. The power level of the 3G cellular interface at different states

IDLE state after another inactivity timer expires. The timers are configured by the network operator, and they are typically very long and are referred to as the long *tail time* (t_{tail}). During this tail time, the radio interface is kept at a high power level even though there is no data transmission.

LTE has similar radio resource control states with 3G. Thus, we generalize the power consumption of 3G/LTE cellular interface into three states: *promotion*, *data transmission* and *tail*. The power in the promotion and tail state are denoted as P_{pro}, and P_{tail}. We differentiate the power for uploads from downloads in data transmission, and denote them as P_{up} and P_{down}, respectively.

C. Task Execution Model

A task can be executed locally on the mobile device or executed on the server through offloading. If task T_i is executed on the mobile device, its execution time is denoted as $t_m(i)$, and its energy consumption is denoted as $E_m(i)$. If T_i is offloaded to the remote server, the whole process consists of three steps.

1) Sending the Data (Input): First, the mobile device sends T_i's input data $d_{in}(i)$ to the remote server via the uplink. Suppose the upload bandwidth of the cellular network is denoted as B_{up}. The transmission time of the input data is $t_{send}(i) = d_{in}(i)/B_{up}$, and the energy consumption of the mobile is $E_{send}(i) = P_{up} \cdot t_{send}(i)$.

2) Executing Task on the Server: After sending the input data to the server, the mobile device is idle waiting for the result. Let $t_{wait}(i)$ denote T_i's execution time on the server (i.e., mobile device's waiting time). Although the CPU of the mobile device is idle, its cellular interface may not move to IDLE state due to the long tail time. If $t_{wait}(i)$ is shorter than t_{tail}, the power level will stay at P_{tail} during this time period. Otherwise, the power level will be P_{tail} for t_{tail}, and then stay at P_{idle} for the remaining time. To summarize, the energy consumption of the mobile device when the task is executed on the server is calculated as

$$E_{wait}(i) = \begin{cases} P_{tail} \cdot t_{wait}(i), & \text{if } t_{wait}(i) \le t_{tail} \\ P_{tail} \cdot t_{tail} + P_{idle} \cdot (t_{wait}(i) - t_{tail}), & \text{otherwise.} \end{cases} \quad (1)$$

3) Receiving the Data (Output): After the server finishes task T_i, it sends back the output data $d_{out}(i)$. If the task execution takes longer than t_{tail}, the cellular interface will be in IDLE state, and there will be a promotion delay, so the time to receive all data is $t_{rcv}(i) = d_{out}(i)/B_{down} + t_{pro}$, where B_{down} is the download bandwidth. Otherwise, the mobile device only consumes energy to download the output data and then $t_{rcv}(i) = d_{out}(i)/B_{down}$. That is,

$$E_{rcv}(i) = \begin{cases} P_{down} \cdot (d_{out}(i)/B_{down}) + P_{pro} \cdot t_{pro}, & \text{if } t_{wait}(i) \ge t_{tail} \\ P_{down} \cdot (d_{out}(i)/B_{down}), & \text{otherwise.} \end{cases} \quad (2)$$

In summary, the energy consumption of a mobile device to offload task T_i is computed as $E_s(i) = E_{send}(i) + E_{wait}(i) + E_{rcv}(i)$, and time to offload T_i is $t_s(i) = t_{send}(i) + t_{wait}(i) + t_{rcv}(i)$. To ensure that our scheme will not increase the response time, task T_i will be executed on the mobile if $t_m(i) \le t_s(i)$. $E_s(i)$ does not include the tail energy after receiving the output or the promotion energy before sending the input, which are affected by the status of adjacent tasks and will be further discussed in the next two sections.

IV. Optimal computation offloading

In this section, we define the problem of computation offloading and introduce an offline algorithm to find the optimal solution. We first propose a solution to reduce the decision states and build a solution graph that enumerates all possible decisions. Then, we transform the problem to a shortest path problem and use Dijkstra's algorithm to find the optimal solution. At last, we discuss how to adapt the algorithm for parallel task scenarios.

A. Problem Statement

Suppose a user runs computationally intensive applications on a mobile device which generate n tasks. The ith task T_i, with input data size $d_{in}(i)$ and output data size $d_{out}(i)$, is generated at time $t_0(i)$. After a task is generated, it is executed either locally on the mobile device or on the remote server. The *computation offloading problem* is to find an offloading solution $D = (l_1, \ldots, l_i, \ldots, l_n)$ which executes all tasks with the minimum amount of energy, where $l_i = 0$ means that T_i should be executed on the mobile device whereas $l_i = 1$ means that T_i should be offloaded to the server. We first assume that tasks are executed sequentially and in the order they are generated, and then discuss how to adapt the proposed approach for parallel tasks in Section IV-E.

When a task is offloaded, the communication between the mobile device and the server may suffer from the long tail problem. This may impact later tasks, which makes it hard to calculate the energy consumption of a later offloaded task. Specifically, if T_i is offloaded to the server, its energy consumption depends on how long the remaining tail is between T_i and the previous offloaded task. Thus, we need to know the last tail (which is generated by the last offloaded task) before task T_i to calculate T_i's offloading cost. As each task

has two options, there are 2^{i-1} possibilities (decision states) to determine the last tail of the previous tasks. Therefore, it is a challenge to deal with the exponentially increasing decision states when making offloading decisions.

B. Reducing the Decision States

The above solution has exponential states because it relies on the decisions of all previous tasks to determine the status of the last tail. Actually, this is not necessary since we can let tasks update and record the status of the last tail. More specifically, to calculate T_i's offloading cost, we need to know T_{i-1}'s last tail by considering the following two cases. First, T_{i-1} is offloaded to the server. As T_{i-1} may be either affected or not affected by a former tail, we need two decision states to represent the two different tails generated by offloading T_{i-1}. Second, T_{i-1} is executed on the mobile. T_{i-1}'s recorded last tail is generated by a task prior to T_{i-1}, which could be T_{i-2}, T_{i-3}, ..., or even T_1. Each of them can be affected or not affected by a former tail when being offloaded, resulting in two decision states for each of them, with a total of $2(i-2)$ states. Since T_1 is not affected by any previous tail, only one decision state (instead of two) is needed to record its generated tail, and then we need $2(i-2)-1$ decision states in the second case. Considering both cases, we need $2(i-1)-1$ decision states to represent all the possibilities of T_{i-1}'s last tail. Then, the decision states increase linearly rather than exponentially.

Formally, we denote a decision state of task T_i as a virtual node $V^i_{L_i,I_i}$, where L_i indicates the offloading decision of T_i:

$$L_i = \begin{cases} S & T_i \text{ is executed on the remote server} \\ M & T_i \text{ is executed on the mobile device} \end{cases}$$

and I_i indicates whether T_i is affected by a previous tail:

$$I_i = \begin{cases} 0 & \text{not affected by a former tail} \\ 1 & \text{affected by a former tail.} \end{cases}$$

The combination of L_i and I_i results in four different types of decision states: $V^i_{S,0}$, $V^i_{S,1}$, $V^i_{M,0}$ and $V^i_{M,1}$.

Task T_i's offloading decision can affect the state of the next task T_{i+1}. For each decision state $V^i_{L_i,I_i}$ of T_i, its *TailEndTime* ($t^i_{L_i,I_i}$) is defined as the end time of a potential tail that could affect the next task. This tail is generated by T_i itself or its previous tasks. Each node needs to compute and maintain its TailEndTime, and different types of nodes have different methods to compute TailEndTime.

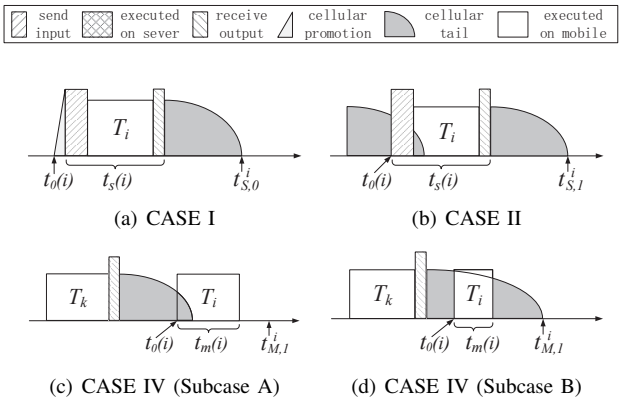
CASE 1 (Fig. 2(a)): $t^i_{S,0} = t_0(i) + t_{pro} + t_s(i) + t_{tail}$.

For $V^i_{S,0}$, task T_i is offloaded to the server, and the cellular interface is in IDLE since $I_i = 0$. It takes a promotion delay (t_{pro}) before sending the input data. As described in Section III-C, $t_s(i) = t_{send}(i) + t_{wait}(i) + t_{rcv}(i)$.

CASE 2 (Fig. 2(b)): $t^i_{S,1} = t_0(i) + t_s(i) + t_{tail}$.

For $V^i_{S,1}$, the difference from Case 1 is that $I_i = 1$, where the cellular interface is affected by a former tail. Then, the mobile device can send the input data to the remote server at time $t_0(i)$.

CASE 3: $t^i_{M,0} = -\infty$.



(a) CASE I (b) CASE II

(c) CASE IV (Subcase A) (d) CASE IV (Subcase B)

Fig. 2. Different cases to compute TailEndTime

For $V^i_{M,0}$, T_i is executed on the mobile device. As $I_i = 0$ indicates the cellular interface is not affected by a former tail, and its state will not change during T_i's execution on the mobile device, the cellular interface should still be in IDLE when T_{i+1} arrives. In this case, $V_{M,0}$ will not affect the next task, and thus we have $t^i_{M,0} = -\infty$.

CASE 4: $t^i_{M,1} = -\infty$ or $t^i_{M,1} = t^{i-1}_{L_{i-1},I_{i-1}}$.

For $V^i_{M,1}$, $I_i = 1$ means that the cellular interface is affected by a former tail at time $t_0(i)$. Although T_i's execution on the mobile device will not affect the state of the cellular interface, the cellular interface can change to IDLE by itself as time goes by. There are two possible cases. In one case as shown in Fig. 2(c), the tail ends and the cellular interface moves to IDLE during T_i's execution on the mobile. In this case, $V^i_{M,1}$ will not affect the next task, thus $t^i_{M,1} = -\infty$. In another case (Fig. 2(d)), T_i ends before the former tail ends. Then the former tail may still affect the next task. T_i has to maintain this information by recording the status of this tail. Although this tail is generated by T_k, its status is recorded and passed on to later tasks including T_{i-1}. Therefore, T_i records the tail by inheriting its parent node's TailEndTime.

C. Building a Solution Graph

After reducing the decision states, we build a solution graph based on which we can find the optimal solution. In the solution graph, the decision states of a task are the nodes. Task T_i's virtual nodes $V^i_{S,0}$, $V^i_{S,1}$, $V^i_{M,0}$ and $V^i_{M,1}$ are at depth $h = i$. We also create two dummy nodes: V_{start} as the source node with depth $h = 0$ and V_{end} as the destination node with depth $h = n+1$. Then, $t_{V_{start}} = -\infty$ and $t_{V_{end}}$ is meaningless.

For edges, we start from V_{start} and employ a top-down approach. For task T_i, we check the corresponding nodes at depth i. For each node $V^i_{L_i,I_i}$, we analyze its possible next state $V^{i+1}_{L_{i+1},I_{i+1}}$ and create an edge between them. As task T_{i+1} can either be executed on the mobile device or on the server no matter where T_i is, L_{i+1} can either be S or M. However, the value of I_{i+1} depends on the TailEndTime of T_i

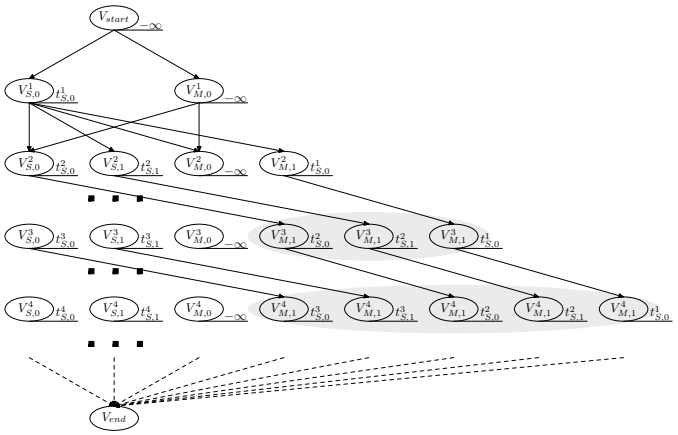


Fig. 3. Solution graph for the offloading problem

and the arrival time of T_{i+1}, as shown in Eq. 3.

$$I_{i+1} = \begin{cases} 0 & \text{if } t_0(i+1) > t^i_{L_i,I_i} \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

An example of such a solution graph is shown in Fig. 3. The TailEndTime of each node is indicated on the line next to the node. Theoretically, each virtual node at depth $h = i$ has four children. However, $V^i_{M,0}$ is an exception and only has two children $V^{i+1}_{S,0}$ and $V^{i+1}_{M,0}$. This is due to $t^i_{M,0} = -\infty$ (in Case 3), implying $V^i_{M,0}$ can not affect task T_{i+1}. As a result, no matter task T_{i+1} is executed on the server or on the mobile, the cellular interface must be in IDLE when T_{i+1} arrives, thus I_{i+1} must be 0.

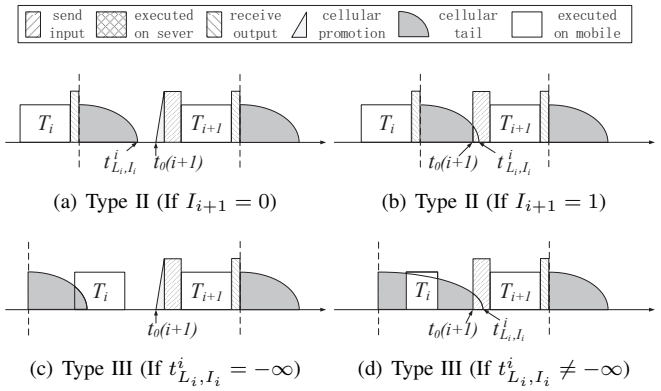
The edge weight from a virtual node $V^i_{L_i,I_i}$ to its child $V^{i+1}_{L_{i+1},I_{i+1}}$, denoted as W^{i+1}_i, is defined as the additional energy consumed by the mobile device to execute T_{i+1}. The computation of this weight is not straightforward, since the additional energy of an offloaded task is related with the previous offloaded task. To solve this problem, we analyze different types of edges, and show that the computation of edge weight only depends on the locations of both task T_i and T_{i+1}, and the temporal relation between them.

TYPE 1: $L_{i+1} = M$

For this type of edge, T_{i+1} is executed on the mobile device. Then, the edge weight W^{i+1}_i is the energy consumed to execute T_{i+1} on the mobile device, which is $E_m(i+1)$.

TYPE 2: $L_i = S$ and $L_{i+1} = S$

$L_i = S$ means that T_i is offloaded to the server, so T_i is the previous offloaded task for T_{i+1}. The additional energy consumed to execute T_{i+1} consists of three parts: tail energy after T_i, promotion energy before T_{i+1} and execution energy of T_{i+1}. The offloading of T_{i+1} to the server may help save some tail energy depending on the decision state of T_{i+1}. If $I_{i+1} = 0$, there is a complete tail as shown in Fig. 4(a). If $I_{i+1} = 1$, T_{i+1} can compensate some tail which is $t^i_{L_i,I_i} - t_0(i+1)$ as shown in Fig. 4(b). For the promotion energy, it depends on the status of T_{i+1}. If $I_{i+1} = 0$, promotion is required. Otherwise, promotion energy can be saved. For the execution energy, $L_{i+1} = S$ means T_{i+1} is executed on the server, so the energy is $E_s(i+1)$. As a result, we can compute the edge weight using Eq. 4.



(a) Type II (If $I_{i+1} = 0$) (b) Type II (If $I_{i+1} = 1$)

(c) Type III (If $t^i_{L_i,I_i} = -\infty$) (d) Type III (If $t^i_{L_i,I_i} \neq -\infty$)

Fig. 4. Different types of edges to compute edge weight

$$\begin{aligned} W^{i+1}_i =& P_{tail} \cdot \left(t_{tail} - I_{i+1} \cdot \left(t^i_{L_i,I_i} - t_0(i+1)\right)\right) \\ &+ (1 - I_{i+1}) \cdot P_{pro} \cdot t_{pro} + E_s(i+1) \end{aligned} \tag{4}$$

TYPE 3: $L_i = M$ and $L_{i+1} = S$

Similar to Type 2, the additional energy consumed to execute T_{i+1} still consists of three parts. To compute the edge weight, we need to know the last tail before T_{i+1}. $L_i = M$ means task T_i is executed on the mobile device, although it cannot be the producer of the last tail, it maintains the end time of the last tail as its TailEndTime. If $t^i_{L_i,I_i} = -\infty$, the last tail has ended before task T_{i+1} starts. In this case, there is a complete tail before T_{i+1} as shown in Fig. 4(c). If $t^i_{L_i,I_i} \neq -\infty$, the offloading of T_{i+1} to the server may help save some tail energy and promotion energy depending on the status of T_{i+1} as shown in Fig. 4(d). We can use Eq. 5 to compute the weight accordingly.

$$W^{i+1}_i = \begin{cases} P_{tail} \cdot t_{tail} + P_{pro} \cdot t_{pro} + E_s(i+1), \\ \qquad\qquad \text{if } t^i_{L_i,I_i} = -\infty \\ P_{tail} \cdot \left(t_{tail} - I_{i+1} \cdot \left(t^i_{L_i,I_i} - t_0(i+1)\right)\right) \\ +(1 - I_{i+1}) \cdot P_{pro} \cdot t_{pro} + E_s(i+1), \\ \qquad\qquad \text{if } t^i_{L_i,I_i} \neq -\infty \end{cases} \tag{5}$$

We notice that $t^i_{L_i,I_i} = -\infty$ must lead to $I_{i+1} = 0$ (by Eq. 3), then Eq. 5 can be transformed and further be merged with Eq. 4. In summary, we can compute the edge weight using Eq. 6 for all types of edges, except that the edge weight from V_n to V_{end} is set to 0.

$$W^{i+1}_i = \begin{cases} E_m(i+1), & \text{if } L_{i+1} = M \\ P_{tail} \cdot \left(t_{tail} - I_{i+1} \cdot \left(t^i_{L_i,I_i} - t_0(i+1)\right)\right) \\ +(1 - I_{i+1}) \cdot P_{pro} \cdot t_{pro} + E_s(i+1), \\ \qquad\qquad \text{if } L_{i+1} = S \end{cases} \tag{6}$$

D. Optimal Offloading Algorithm

We transform the optimal offloading problem to finding the shortest path in terms of energy consumption from V_{start} to V_{end} in the solution graph.

During the construction of the solution graph, we will at most generate one $V^i_{S,0}$, one $V^i_{S,1}$ and one $V^i_{M,0}$ for task T_i. For $V^i_{M,1}$, it inherits its parent nodes' TailEndTime (in Case 4). If it has multiple parent nodes, multiple $V^i_{M,1}$ nodes with different TailEndTime are generated. Therefore, the number of nodes for each task increases as shown in Fig. 3. Task T_1 has only two nodes with subscript $I_1 = 0$ as the first task must not be affected by a former tail. T_2 has four nodes and one of them is $V^2_{M,1}$. T_3 has three $V^3_{M,1}$ nodes as there are three parent nodes at depth 2. According to this pattern, the number of nodes at depth i is $2i$. Then, the solution graph has $|V| = O(n^2)$, and $|E| = O(n^2)$.

We use Dijkstra's algorithm to compute the shortest path. Based on the scale of the solution graph, the algorithm can compute the shortest path in polynomial time $O(n^2 \log n)$. To get the optimal offloading solution, we back track from V_{end}. If the shortest path traverses through node with $L_i = S$ at depth i, $l_i = 1$. Otherwise, $l_i = 0$. Next, we prove that the solution we get is indeed the optimal solution.

Theorem 1: The shortest path of the solution graph represents the optimal solution of the computation offloading problem.

Proof: Let D denote the solution constructed based on the shortest path p, and $E(D)$ is the overall energy consumption. Suppose to the contrary that D is not the optimal solution, then there must exist a solution $D' = (l'_1, \ldots, l'_i, \ldots, l'_n) \neq D$ such that $E(D') < E(D)$.

We can map D' to a path on the solution graph G as follows. For T_i with $l'_i = 1$, it is mapped to node $V^i_{S,0}$ if the cellular interface is in IDLE state when T_i arrives, otherwise it is mapped to $V^i_{S,1}$. For T_i with $l'_i = 0$, it is mapped to node $V^i_{M,0}$ if the cellular interface is in IDLE state. Otherwise, we find the maximum j such that $l'_j = 1$ and $j < i$. If T_j has been mapped to $V^j_{S,0}$, T_i is mapped to $V^i_{M,1}$ with $t^i_{M,1} = t^j_{S,0}$; otherwise, T_i is mapped to $V^i_{M,1}$ with $t^i_{M,1} = t^j_{S,1}$. As shown in Fig. 3, there are $2(i-1)-1$ $V^i_{M,1}$ nodes with different TailEndTime: two with $t^k_{S,0}$ and $t^k_{S,1}$ inherited from every $T_k (1 < k < i)$ and one with $t^1_{S,0}$ inherited from T_1. Thus, T_i can be definitely mapped to the corresponding node. Considering the construction process of G, there must exist a path p' from V_{start} to V_{end} that traverses all and only the mapped nodes, and its weight is $W(p')$.

Next, we need to prove $W(p') = E(D')$. We divide the edges of p' into two sets: $S_1 = \{(V^i, V^{i+1}) | L_{i+1} = M\}$ and $S_2 = \{(V^i, V^{i+1}) | L_{i+1} = S\}$, with weight $W(S_1)$ and $W(S_2)$, respectively. $E(D')$ also consists of two part: $E(D')_1$ and $E(D')_2$, which are the overall energy of tasks executed on the mobile and the server, respectively. According to Eq. 6 (case $L_{i+1} = M$), $E(D')_1 = \sum_{l'_i=0} E_m(i) = W(S_1)$. Then we use a sequence to represent all tasks that are executed on the server, and use T_j and T_{i+1} to denote any two consecutive tasks. Since the long tail generated by T_j is maintained by T_i as $t^i_{L_i,I_i}$, we can use Eq. 6 (case $L_{i+1} = S$) to calculate the energy to execute T_{i+1} and the tail energy between T_j and T_{i+1}. Summing up the energy of all $\sum_{i=1}^{n} l'_i$ tasks, we get $E(D')_2 = W(S_2)$.

As solution D represents the shortest path on G, we have $E(D) = W(p) \leq W(p') = E(D')$. A contradiction. ■

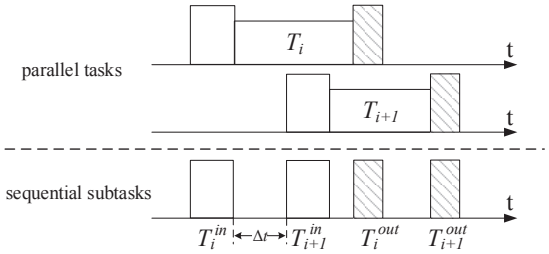


Fig. 5. Converting parallel tasks to sequential subtasks

E. Parallel Tasks

The above solution assumes that the task executions do not overlap with each other. In some cases, several tasks may arrive at the same time. For example in Fig. 5, suppose the cellular interface receives task T_i immediately followed by another task T_{i+1}, and both tasks are offloaded to the server. After sending T_i's input, the cellular interface continues to send T_{i+1}'s input while T_i is being executed at the server. When parallel tasks are allowed, the task execution model has to be modified. Next, we explain how to adapt our optimal offloading algorithm to find the optimal decisions when parallel tasks exist.

In our task execution model (Section III-C), the energy to offload task T_i consists of the energy to send the input and output data, and the tail energy between them. When tasks are sequential, this tail energy only depends on the execution time of the offloaded task on the server. However, when tasks are overlapped, the calculation becomes complicated since there may be multiple data transmissions between T_i's input and output data transmissions. To calculate the tail energy of each data transmission, we modify the model by considering the input and output data transmission of each task as two separate subtasks. As shown in Fig. 5, we use subtask T^{in}_i and subtask T^{out}_i to represent task T_i's input and output data transmission, respectively. In this way, the original parallel tasks are converted to sequential subtasks. When subtasks are also overlapped ($\Delta t < 0$), there is no tail energy between them.

Using the modified model, the original problem to find optimal offloading decisions for n tasks transforms to finding the optimal solution for $2n$ sequential subtasks. With the constraint that T^{in}_i and T^{out}_i must have the same offloading decision, we can then adapt our optimal offloading algorithm to solve the problem. We will not elaborate the details in this paper due to space limitation.

V. Online computation offloading

The optimal computation offloading algorithm can minimize the energy consumption. However, it requires the complete knowledge of all tasks such as task intervals and task execution profiles (i.e., the energy consumed to execute the task on the

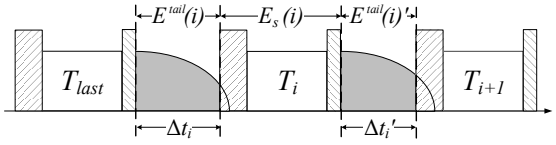


Fig. 6. Energy consumption to offload task T_i

smartphone and the server). In this section, we relax these requirements and present an online computation offloading algorithm.

A. Online Offloading Algorithm

Intuitively, a computationally intensive task should be offloaded under two scenarios. Suppose the last offloaded task T_{last} before T_i happened at t_1, T_i arrives at t_2, and T_{i+1} will arrive at t_3. First, if $t_2 - t_1$ is small, sending T_i to the server can cut the long tail generated by T_{last}. Second, if $t_3 - t_2$ is small, the long tail generated by offloading T_i may also be cut if T_{i+1} is offloaded too. The first scenario is easy to identify as the mobile device has all the information of past tasks. However, the second scenario is hard to identify and we have to rely on predictions. If we can predict that T_{i+1} will arrive soon after T_i (will be quantified later), then offloading task T_i can save energy. In the rest of this section, we introduce our prediction method in the online algorithm.

The basic idea of our online algorithm is to offload a task when there is a high probability that offloading the task consumes less energy than executing it on the mobile, i.e.,

$$P(E_s^{total}(i) < E_m^{total}(i)) \geq \theta \tag{7}$$

where θ is a system parameter. $E_s^{total}(i)$ and $E_m^{total}(i)$ denote the total energy consumption to offload task T_i and to execute it on the mobile device, respectively. The superscript "total" means that the energy also includes the tail energy related to T_i. $E_m^{total}(i) = P_{tail} \cdot t_{tail} + E_m(i)$, which includes the energy of a complete tail as $E^{tail}(i)$. As shown in Fig. 6, $E^{tail}(i)$ is the tail energy between T_{last} and T_i, and $E^{tail}(i)'$ is the tail energy between T_i and T_{i+1}. $E^{tail}(i)'$ is 0 since executing T_i on the mobile will not generate a long tail between T_i and T_{i+1}. $E_s^{total}(i)$ is the sum of $E^{tail}(i)$, the server execution energy $E_s(i)$, and $E^{tail}(i)'$.

Since the mobile device has already made offloading decisions for the past tasks, $E^{tail}(i)$ can be calculated easily. Letting Δt_i denote the time interval between T_i and T_{last}, $E^{tail}(i)$ can be calculated as follows. If Δt_i is smaller than t_{tail}, there is only partial tail energy $P_{tail} \cdot \Delta t_i$; Otherwise, offloading T_i will consume extra promotion energy and a complete tail energy.

The calculation of $E^{tail}(i)'$ depends on the time interval $\Delta t_i'$ between T_i and T_{i+1}. Since the mobile device has no knowledge of future tasks, we can only get $E^{tail}(i)'$ based on prediction. Let random variable T denote the task arrival interval, then the offloading requirement given by Eq. 7 can be specified as

$$P(T < \Delta t_i') = F(\Delta t_i') \geq \theta. \tag{8}$$

Algorithm 1: The online algorithm

Data: Task set
Result: Decision sequence D
Last offloaded task $T_{last} \leftarrow$ *null*;
Last transfer end time $t_{last} \leftarrow -\infty$;
for $i \leftarrow 1$ **to** n **do**
  $\Delta t_i \leftarrow a_0(i) - t_{last}$;
  if $\Delta t_i < t_{tail}$ **then**
    $E^{tail}(i) \leftarrow P_{tail} \cdot \Delta t_i$;
  else
    $E^{tail}(i) \leftarrow P_{pro} \cdot t_{pro} + P_{tail} \cdot t_{tail}$;
  end
  $E_m^{total}(i) \leftarrow P_{tail} \cdot t_{tail} + E_m(i)$;
  $\Delta t_i' \leftarrow \frac{E_m^{total}(i) - (E^{tail}(i) + E_s(i))}{P_{tail}}$;
  $P(\Delta t_i') \leftarrow$ Eq. 8 ;
  if $P(\Delta t_i') \geq \theta$ **then**
    offload T_i;
    $l_i = 1$;
    update $T_{last} \leftarrow T_i$;
    update t_{last} ;
  else
    execute T_i on mobile device;
    $l_i = 0$;
  end
end

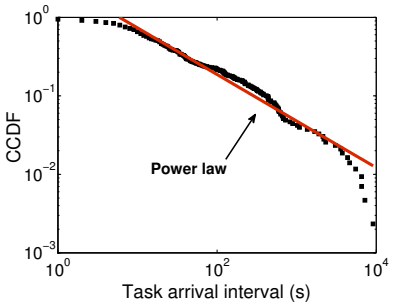


Fig. 7. CCDF of the task arrival interval

$P(T < \Delta t_i')$ is the probability that there is a task within $\Delta t_i'$ time from task T_i, where $\Delta t_i'$ is obtained by solving the equation $E_s^{total}(i) = E_m^{total}(i)$. $F(t)$ is the CDF of the task arrival interval T which is obtained through experiments. A complete description of the online offloading algorithm is shown in Algorithm 1.

B. Task Arrival Interval

We distributed five Android smartphones with our Optical Character Recognition (OCR) application installed to users, and collected one week of trace which has 528 tasks. Fig. 7 shows the CCDF (Complementary CDF) of the task arrival interval. As can be seen, the CCDF values exhibit linear decay which suggests power-law characteristics. By definition, it means that many task arrival intervals are short and few are long. The CDF of the task arrival interval T is given by

$$F(t) = \begin{cases} 1 - (\frac{t_{min}}{t})^{\beta}, & \text{if } t \geq t_{min} \\ 0, & \text{if } t < t_{min} \end{cases}$$

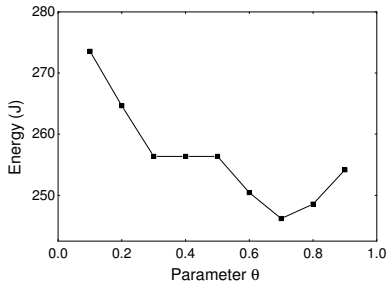


Fig. 8. The effects of θ

where t_{min} is the minimum possible value of T, and β is a positive parameter.

C. Parameter Setting

Parameter θ directly affects the decisions of our online algorithm. We perform an experiment to help select θ. We choose 20 tasks from the trace and run our online algorithm on a smartphone with different θ. For each θ, the smartphone executes tasks locally and offloads to the server according to the decisions made by the online algorithm. We measure the energy consumption of the phone in each round and plot the result in Fig. 8. We find that setting θ to 70% can get the best performance in terms of energy consumption, thus, we use it as the default value in our algorithm.

VI. Performance Evaluations

To evaluate the performance of the computation offloading algorithms, we have implemented the proposed algorithms on smartphones and compared their performance to existing solutions through experiments and trace-driven simulations.

A. Experimental Setup

We have implemented an OCR application based on Tesseract library [22] on Android-based smartphones and on the remote server. The OCR application provides automatic conversion of photographed images of printed text into machine-readable text. We also implemented the online offloading algorithm on smartphones. The mobile device is a Samsung Galaxy S III (GS3) smartphone running Android 4.0.2. The server is a dual-core desktop with a 2.3GHZ CPU and a 6GB of RAM running Windows 7.

TABLE I
State machine parameters of different networks

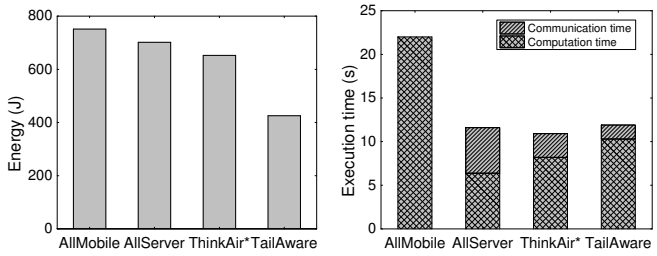
Network	State	Power(mW)	Duration(s)
3G(AT&T)	Promotion	1102.2±40.4	0.9±0.1
	Tail	1414.8±163.4	8.0±0.1
LTE(AT&T)	Promotion	1214.8±24.3	0.25±0.4
	Tail	1455.9±26.4	11.5±0.56

TABLE II
Data transmission parameters of different networks

Network	State	Power(mW)	Throughput(Mb/s)
3G(AT&T)	Download	1408.3±96.4	2.3±0.6
	Upload	1432.6±105.7	1.4±0.7
LTE(AT&T)	Download	1827.5±30.1	51.11±16.9
	Upload	1865.8±25.6	17.9±5.1
Phone(GS3)	Idle	679.7±9.7	-

TABLE III
Parameters of OCR tasks

Average data size(KB)	Input	1010.4
	Output	53.7
Average execution time(s)	On mobile	21.9
	On server	6.3



(a) Energy (b) Execution time

Fig. 9. Energy and execution time of different offloading algorithms

To setup the parameters, we measured the power consumption and the throughput in 3G and LTE when the cellular interface is at different states. To measure the power consumption, we use the Monsoon Power Monitor [23] to provide constant power to the smartphone instead of using battery. The results are shown in Table I and Table II.

To get the task execution time and the energy model defined in Section III-C, we run the application multiple times with different input data and measure the task execution time and the power level of the smartphone. The results are used to predict the task execution time based on the input data. A simple linear model is used to predict the energy consumption as a function of the task execution time.

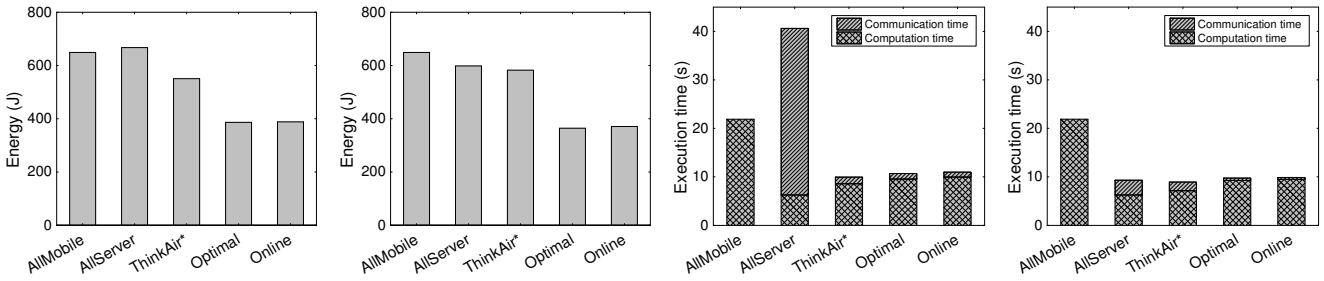
Our algorithm is compared with the following approaches:

- **AllMobile**: all tasks are executed on the smartphone.
- **AllServer**: all tasks are executed on the remote server.
- **ThinkAir*** [15]: the decision of offloading is based on user specified policy. Among the four policies in ThinkAir, the "Energy" policy has the same goal as our algorithm and is used for comparison. It compares the energy consumption of running the computation locally and that of offloading to the cloud, and only offloads the computation when energy can be saved. However, ThinkAir does not consider the long tail problem.

The metrics used for comparison are energy consumption and *execution time*. If a task is offloaded, the execution time consists of communication time and computation time.

B. Experimental Results

In this section, we compare our algorithms with others using real experiments. In the experiments, the WiFi interface of the smartphone is turned off and the phone works under LTE. 40 OCR tasks are used in the evaluation, and the parameters of the



(a) Energy-3G (b) Energy-LTE (c) Execution time-3G (d) Execution time-LTE

Fig. 10. Performance comparison of different offloading algorithms in different cellular networks

tasks are shown in Table III. The task arrival time is recorded when the experiment is run and then replayed when other algorithms are used. For example, in AllMobile, the timestamp of the task arrival time is recorded. When AllServer is run, we still use the same task arrival time.

Fig. 9 shows the experimental results. The experimental results show that our offloading algorithm ("TailAware") can save more energy than other approaches, which is about 43% compared to AllMobile, 39% compared to AllServer, and 34% compared to ThinkAir*. AllServer only consumes 7% less energy than AllMobile, which indicates that the energy spent for offloading is almost comparable to the energy saved by offloading. For ThinkAir*, since it does not consider the long tail problem, it may offload a task even though the actual energy consumption including the tail energy is more than local execution, and hence consumes more energy than ours. Note that if fast dormancy is used, more tasks will be offloaded in our algorithm. This is because fast dormancy can reduce the tail energy and then some tasks used to be executed on the mobile to avoid heavy communication cost will become worth offloading.

Fig. 9(b) compares the execution time, which includes the communication time and the computation time. For AllMobile, the communication time is 0 since its computation is not offloaded. For AllServer, although it has low computation time, its communication time is much higher than others. The goal of our algorithm is to minimize the energy, but it also achieves a balance between communication time and computation time. From the figure, we can see that our algorithm reduces the execution time by 46% compared to AllMobile. The execution time of our algorithm is similar to AllServer and ThinkAir*, but our algorithm can save much more energy.

C. Trace-Driven Simulations

In our experiment, we also collected the task traces, which are used for evaluating the performance of our algorithms. Such trace-driven simulations will give us more flexibility to study the effects of other parameters on performance. In the simulations, we compare AllMobile, AllServer, ThinkAir*, and both variants (Optimal and Online) of our TailAware offloading algorithm.

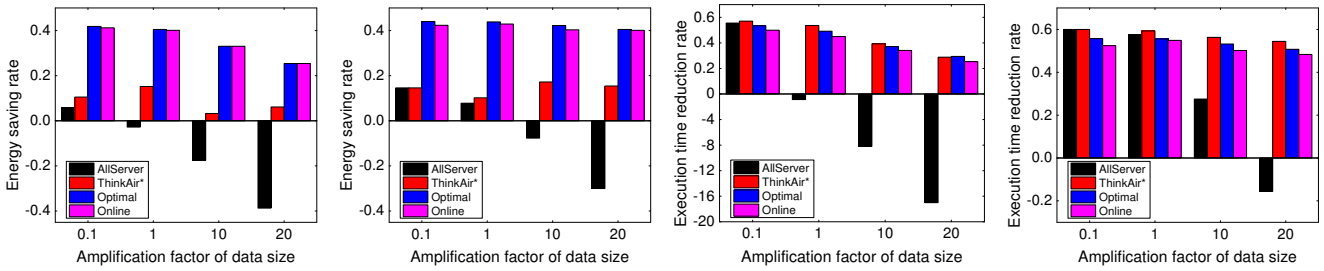
1) Comparisons on Energy Consumption and Execution Time: The power consumption and execution time of different approaches in 3G and LTE networks are shown in Fig. 10. Overall, our offloading algorithms can save at least 43% of energy in 3G and 40% in LTE networks, and reduce the execution time by 50% in 3G and 55% in LTE networks compared to that without computation offloading.

For power consumption, by comparing Fig. 10(a) and Fig. 10(b), we have the following observations. First, our algorithms have better performance in LTE than in 3G network. This is because LTE has much higher throughput and thus can reduce the time in high power state for data transmission. As AllMobile does not use the cellular interface, the energy consumption does not change in different networks. Second, ThinkAir* outperforms AllMobile and Allserver. Compared to ThinkAir*, our algorithms consume 30% less energy in 3G networks, and 38% less energy in LTE networks. Third, the online offloading algorithm can achieve similar energy saving as the optimal.

Fig. 10(c) compares the execution time under 3G. As can be seen, AllServer is the worst. This is because 3G has limited bandwidth and AllServer spends much longer time on offloading the tasks to the server. This result is changed when LTE is used, as shown in Fig. 10(d). The reason is that LTE has much higher throughput, and then AllServer spends less time to offload the tasks to the server compared to AllMobile. From the figures, we can also see that our algorithms have similar execution time compared to ThinkAir*.

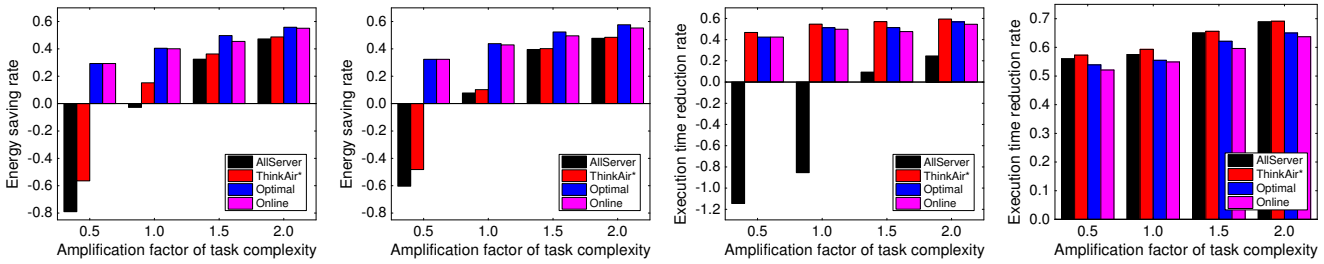
2) Impact of Task Data Size: The input and output data size of a task (task data size) will affect the energy consumed during offloading. In this section, we change the input and output data size by the same factor, and evaluate its effects on performance.

Fig. 11(a) and Fig. 11(b) show the energy saving rate of different approaches compared to AllMoblie. As can be seen, with the increase of data size, the energy saving drops quickly for AllServer, whereas our algorithms can still achieve good performance. In both 3G and LTE, with the increase of data size, AllServer even consumes more energy than AllMobile. This indicates that the computational energy saved by offloading is less than the energy consumed by data transmission as the task data size increases. Then, always offloading tasks to the server (AllServer) becomes a bad choice. Since LTE has much higher bandwidth than 3G, the impact of increasing the



(a) Energy-3G (b) Energy-LTE (c) Execution time-3G (d) Execution time-LTE

Fig. 11. Performance of offloading algorithms when the input/output data size of the task (task data size) changes



(a) Energy-3G (b) Energy-LTE (c) Execution time-3G (d) Execution time-LTE

Fig. 12. Performance of offloading algorithms when the task execution time (task complexity) changes

task data size in LTE is less obvious compared to 3G.

For execution time, as shown in Fig. 11(c) and Fig. 11(d), our algorithms outperform AllServer in 3G which significantly increases the execution time as the task data size increases. While in LTE networks, the increase of the task data size has less impact on the transmission time, thus the execution time reduction rate remains relatively high for all approaches except AllServer. Compared to ThinkAir*, our algorithms can achieve similar reduction rate in all cases.

3) Impact of Task Complexity: The complexity of a task affects the computation time at the mobile and the server, and hence the execution time and the energy consumption. For a more complex task, the advantage to use a fast server will be more obvious. In this section, we change the task complexity for the mobile and the server by the same factor and evaluate its effects on performance.

Fig.12(a) and Fig.12(b) show the energy saving rates of different approaches compared to AllMoblie. When the task complexity increases, the energy saving rate also increases since offloading the task will reduce more computation time. However, the advantage of our algorithms over AllServer and ThinkAir* drops as the task complexity increases. This is because the energy saved for computation gradually dominates the overall energy saving as the task complexity increases, and then our algorithms' efforts on energy saving of data transmission including the tail energy become less obvious. When the task complexity is reduced by half (0.5 in x-axis), AllServer and ThinkAir* even consume more energy than AllMobile. This is because at this time the tasks are too simple to be offloaded, and the energy saved in computation is not as much as the energy spent on data transmission. For execution time, since offloading can reduce more computation time as the task complexity increases, the performance also becomes better as shown in Fig.12(c) and Fig.12(d).

In summary, our offloading algorithms can save energy and reduce the execution time when the task data size is small and when the task complexity increases. Also, offloading with LTE can achieve better performance than 3G.

VII. Conclusion

In this paper, we proposed energy-efficient computation offloading algorithms for cellular networks. We formally analyzed the effect of the long tail problem when making offloading decisions and proposed a novel solution to minimize the energy consumption of mobile devices. We formulated the offloading problem as a shortest path problem and used Dijkstra's algorithm to find the optimal solution. We also proposed an online offloading algorithm and have implemented it on Android-based smartphones. Experimental results show that our offloading algorithm can save about 35% of the energy. Trace-driven simulation results show that the proposed offloading algorithms are more effective for complex tasks with small data size, and LTE is a better choice for computation offloading than 3G.

Acknowledgment

We would like to thank our shepherd Joshua Reich and the anonymous reviewers for their insightful comments and helpful suggestions. This work was supported in part by the National Science Foundation (NSF) under grant CNS-1218597, CNS-1526425, and CNS-1421578.

REFERENCES

[1] 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.

[2] 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 ACM International Conference on Mobile Systems, Applications, and Services*, 2010.

[3] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of ACM Conference on Computer Systems*, 2011.

[4] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *Proceedings of USENIX Conference on Hot Topics in Operating Systems*, 2009.

[5] 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.

[6] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proceedings of IEEE International Conference on Computer Communications*, 2012.

[7] B. Zhao, Q. Zheng, G. Cao, and S. Addepalli, "Energy-aware web browsing in 3g based smartphones," in *Proceedings of IEEE International Conference on Distributed Computing Systems*, 2013.

[8] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of ACM SIGCOMM Conference on Internet Measurement Conference*, 2009.

[9] W. Hu and G. Cao, "Quality-aware traffic offloading in wireless networks," in *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2014.

[10] ——, "Energy optimization through traffic aggregation in wireless networks," in *Proceedings of IEEE International Conference on Computer Communications*, 2014.

[11] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services*, 2012.

[12] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li, "Ready, set, go: Coalesced offloading from mobile devices to the cloud," in *Proceedings of IEEE International Conference on Computer Communications*, 2014.

[13] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010.

[14] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[15] 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 *Proceedings of IEEE International Conference on Computer Communications*, 2012.

[16] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading inference for delivering applications in pervasive computing environments," in *Proceedings of IEEE International Conference on Pervasive Computing and Communications*, 2003.

[17] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, "Using bandwidth data to make computation offloading decisions," in *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, 2008.

[18] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services*, 2011.

[19] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," in *Proceedings of IEEE International Conference on Computer Communications*, 2013.

[20] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "Characterizing radio resource allocation for 3g networks," in *Proceedings of ACM SIGCOMM Conference on Internet Measurement*, 2010.

[21] P. K. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D. Arora, V. N. Padmanabhan, and G. Varghese, "Radiojockey: mining program execution to optimize cellular radio usage," in *Proceedings of ACM International Conference on Mobile Computing and Networking*, 2012.

[22] "Tesseract OCR," https://code.google.com/p/tesseract-ocr/.

[23] "Monsoon Power Monitor," http://www.msoon.com/.