Ready, Set, Go: Coalesced Offloading from Mobile Devices to the Cloud

Liyao Xiang[1], Shiwen Ye[1], Yuan Feng[2], Baochun Li[1], and Bo Li[3]

[1]Department of Electrical and Computer Engineering, University of Toronto
[2]Department of Computing, Hong Kong Polytechnic University
[3]Department of Computer Science, Hong Kong University of Science and Technology

***Abstract*—With an abundance of computing resources, cloud computing systems have been widely used to elastically *offload* the execution of computation-intensive applications on mobile devices, leading to performance gains and better power efficiency. However, existing works have so far focused on one application only, and multiple applications are not coordinated when sending their offloading requests to the cloud. In this paper, we propose the new technique of *coalesced offloading*, which exploits the potential for multiple applications to coordinate their offloading requests with the objective of saving additional energy on mobile devices. The intuition is that, by sending these requests in "bundles," the period of time that the network interface stays in the high-power state can be reduced. We present two online algorithms, collectively referred to as *Ready, Set, Go* (RSG), that make near-optimal decisions on how offloading requests from multiple applications are to be best coalesced. We show, both analytically and experimentally using actual smartphones, that RSG is able to achieve additional energy savings while maintaining satisfactory performance.**

I. Introduction

Heralded as a primary feature in mobile cloud computing, *code offloading* from mobile devices to the cloud has received a substantial amount of research attention in the recent literature. The concept of code offloading is intuitively simple: with an abundance of computing power in the cloud computing infrastructure and a keen awareness of power efficiency on mobile devices, it is natural to offload a portion of the computational requests within computationally intensive mobile applications. With its roots dating back to the notion of *thin clients* in the 1990s, code offloading may be instrumental in a wide variety of mobile applications, from natural language processing (*e.g.,* Apple's Siri) to augmented reality.

Code offloading can be performed at the granularity level of thread execution [1], [2], method invocation [3], and even full VM migration [4]. Either way, offloading requests have been well planned, with the optimization objective of gaining better application performance and energy efficiency. To achieve the objective, a typical solution includes a profiler on the mobile device that collects runtime statistics of the mobile application, as well as a solver that partitions the computation in a way that optimizes energy consumption or application performance.

However, existing works have so far focused on one application only. In reality, mainstream mobile operating systems support multitasking, with multiple applications running simultaneously on a mobile device. Particularly, there may be several services running in the background while one or two applications running on screen. A user may ask Siri (or Google Now) about a location, viewing the augumented reality street view on her phone, while in the background downloading a cloud-sourced video over 3G or 4G mobile networks at the same time. When multiple applications send their offloading requests to the cloud independently without any coordination, the cellular or Wi-Fi network interface needs to be activated to transmit these requests, entering the high-power state at arbitrary times. This may potentially consume more energy: once a network interface enters the high-power state, it lingers in this state for a period of time, usually seconds, after completing the transmission of all the existing requests [5]. The amount of energy the network interface consumes in the high-power state before it enters stand-by again, referred to as the *tail energy*, is proportional to the length of time the interface stays in this state (referred to as the *tail time*).

In this paper, we propose the concept of *coalesced offloading*, which seeks to achieve additional energy savings by exploiting the potential for multiple mobile applications to coordinate their code offloading requests to the cloud. Coalesced offloading realizes the intuition that, by sending code offloading requests in "bundles," the period of time that the network interface stays in the high-power state can be reduced, thus saving additional energy. Our proposed technique of coalesced offloading is inspired by *timer coalescing*, used in the kernel of Mac OS X 10.9 Mavericks, that improves the energy efficiency by deferring and shifting computation tasks from multiple applications to the same time interval. To our knowledge, our work represents the first attempt to improve power efficiency by bundling offloading requests from multiple applications in a coalesced fashion.

Since bundling offloading requests may incur additional offloading delays, we choose to formulate the problem of coalesced offloading as a joint optimization problem, with both the energy cost and the response time considered. The highlight of our original contributions is the design of two online algorithms, collectively referred to as *Ready, Set, Go* (RSG), that are designed to solve our optimization problem. As the benchmark for evaluating RSG, we first study an offline algorithm that computes the optimal solution with a

978-1-4799-3360-0/14/$31.00

time complexity of $O(n^2)$, with the impractical assumption that the exact arrival times of future requests from all the applications are known *a priori*. Without any knowledge of upcoming offloading requests beforehand, our deterministic online algorithm is 2-competitive against the optimal offline algorithm, and our randomized online algorithm is $e/(e-1)$-competitive (1.58-competitive). We analytically show that both online algorithms achieve the best possible competitive ratios in their respective cases. Our online algorithms are simple enough to implement: using both simulations and our real-world implementation on the iOS platform, we show that the RSG online algorithm is able to realize an additional energy saving of up to 20% for the deterministic case and 27% for the randomized case with a variety of offloading request patterns.

The remainder of this paper is organized as follows. In Sec. II, we motivate the concept of coalesced offloading, and then formulate an optimization problem that considers both the energy cost and performance. In Sec. III discuss an optimal offline algorithm to solve the problem. In Sec. IV, we propose and analyze both the deterministic and the randomized algorithm in RSG. In Sec. V, we evaluate RSG with both simulations and our real-world implementation. Finally, we discuss our contributions in the context of related work in Sec. VI, and conclude the paper in Sec. VII.

II. Coalesced Offloading: Motivation and Problem Formulation

In this section, we first motivate the notion of coalesced offloading, and then formally formulate the optimization problem of making optimal offloading decisions, considering both the energy cost and application performance.

A. Motivation

With current code offloading techniques, if a portion of the application code (*e.g.,* a method invocation or a thread) is to be offloaded to the cloud, an *offloading request* will be generated, and the cellular or Wi-Fi network interface on the mobile device will be activated, incurring a small ramp-up energy cost, such as the WiFi association overhead. After the completion of transmitting each request, the interface will not immediately switch to the low-power state. Instead, it remains at the high-power state for tens of seconds — an inactive period referred to as the *tail time* [5], as shown in Fig. 1 (a). If there is another request coming in during the tail time, the inactivity timer will be reset, and the interface will stay at the high-power state until the end of the transmission, plus another period of the tail time if there are no further successive requests. The tail time phenomenon is especially critical with the 3G interface, which consumes nearly 60% of the total energy consumption [5].

As an important insight that we explore in this paper, the tail time phenomenon can be alleviated if we *bundle* the offloading requests into small batches, and handle them all together. This reduces the energy consumption as the wireless network interface on the mobile device is activated for fewer times, and a shorter tail time is incurred. Naturally, a single application may not have frequent successive requests for code offloading;

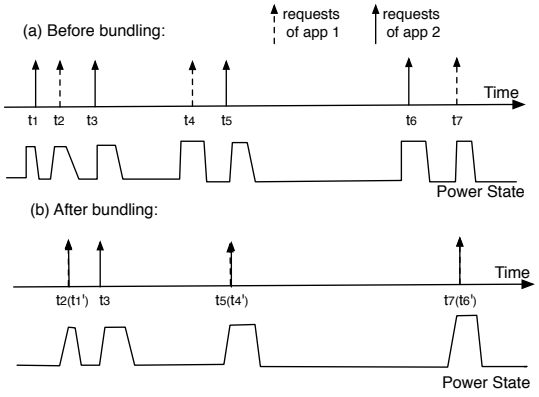


Fig. 1. The benefits of coalesced offloading.

we focus on the abundant request bundling opportunities that exist when we consider the offloading requests from multiple applications running on the device simultaneously. As the example in Fig. 1 (b) shows, three bundles can be formed when offloading requests for two applications are considered at the same time, which lead to a reduced period of time for the network interface to stay in the high-power state, as compared to handling each of them independently without any coordination. Such request bundling from multiple applications is formally referred to as *coalesced offloading* in this paper. It requires all offloading requests to be *granted* by an OS-level coalesced offloading framework, possibly with a delay, before application code is actually offloaded to the cloud.

B. The Coalesced Offloading Problem

While coalesced offloading is able to reduce energy costs, request bundling requires the subsequent offloading requests to wait for a period of time for the next batch to be handled, which results in additional offloading delays and may adversely affect the application performance. The main challenge of coalesced offloading is balancing the tradeoff between the energy cost and the application performance. If requests are bundled more aggressively, less energy costs are incurred as a shorter period of time is spent in the high-power state for code offloading. However, withholding the offloading requests will inevitably cause longer offloading delays. On the other hand, sending offloading requests in a more scattered manner can maintain the high performance of applications, but will incur a longer period of time in the high-power state, causing more energy to be consumed. To find the "sweet spot" in such an inherent tradeoff between energy savings and application performance, we formulate the problem of coalesced offloading as a joint optimization problem, considering both the energy cost and application performance in the objective function.

We assume that there are M applications, $1, 2, \ldots, m$, running on the mobile device, and each application generates multiple offloading requests during their runtime based on their own profiler and solver. Let $a_1, a_2, \ldots$ be the *arrival time sequence* of the offloading requests across all the applications, and $g_1, g_2, \ldots$ be the *granting time sequence*, each element of it representing one transmission from mobile device to the cloud. Notice that multiple requests can be bundled and

granted in one transmission. The granting time directly determines the transitioning time from the low to the high power state. The device transitions from the high to the low power state only when the network has been inactive for the length of tail time. That is to say, the subsequent transmission occurs at least tail time after the preceeding transmission. We use the sequence $t_1, t_2, \ldots$ and $s_1, s_2, \ldots$ to respectively denote such *transition time sequence* when the wireless interface enters the high-power state from the low-power state and the inverse. Let T be the duration of the *tail time* after the completion of transmission. Since the duration of a request transmission is a few orders of magnitude shorter than the length of the tail time (in the order of seconds), we assume that all request transmissions are completed instantaneously.

Fig. 2 shows an illustrative example of our model. The offloading requests arrival time sequence is $a_1, a_2, \ldots, a_9$, and the granting time sequence is $g_1, g_2, \ldots, g_5$. As we can see, two offloading requests generated at time a_1 and a_2 are delayed to be transmitted at g_1, the arrival time of the third request a_3, and the network interface goes into the high-power state. Since the high-power state remains for at least T, requests generated at a_4 and a_5 are transmitted immediately. The network interface transits to the low-power state after idling for time T, and enters the high-power state again at the next transmission time g_4. In a nutshell, we seek to formulate the problem to find the optimal solution of the granting time sequence $g_1, g_2, \ldots$ to determine when the wireless interface of the mobile device should stay at the high-power state for transmitting offloading requests, such that a combined interest in both the energy cost and the application performance is optimized.

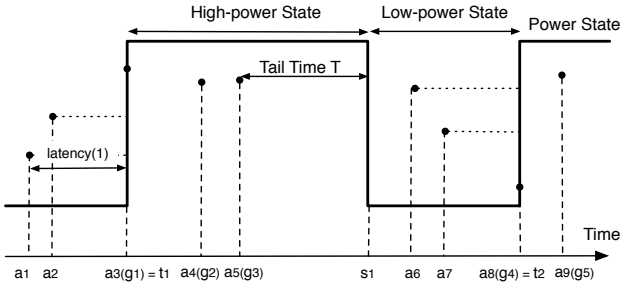


Fig. 2. The coalesced offloading problem: an illustrative example.

Since the actual energy cost is nearly linear to the duration that the network interface stays at the high-power state, in our problem formulation, we use that time duration to represent the energy cost.

Observation 1: If a transmission occurs at g_i when the network interface is in the high-power state, the energy cost is $g_i - g_{i-1}$. If g_i occurs when the network interface is in the low-power state, the energy cost can be considered as T.

To be more specific, when the transmission g_i occurs when the interface is in the high-power state, it extends that state for a period of $g_i - g_{i-1}$. If g_i occurs during the low-power state, it contributes the tail time T to the energy cost. If the duration of $g_i - g_{i-1} > T$, the high-power state will expire before g_i, so that the g_i occurs in the low-power state. Thus, the energy cost for one transmission is $\min\{g_i - g_{i-1}, T\}$. The joint optimization problem of coalesced offloading can be formulated as follows:

$$\min f_{\text{cost}} = \sum_j \min\{g_j - g_{j-1}, T\} + \alpha \sum_j \sum_{\substack{a_i \ s.t. \\ g_{j-1} \le a_i \le g_j}} (g_j - a_i), \tag{1}$$

In the objective function, the first term represents the energy cost while the second term denotes the total latencies as offloading requests are postponed by the coalesced offloading framework. α is introduced to combine the two objectives, and to balance the conflicting interests between minimizing the energy costs and minimizing the total latencies for granting the offloading requests.

At first glance, our formulated problem is similar to the dynamic TCP acknowledgment problem [6]. The dynamic TCP acknowledgment problem discusses the scenario when a number of subsequent messages are to be acknowledged, whether we should acknowledge each individual message immediately upon receiving it, or acknowledge multiple messages with a single acknowledgment packet. While we amortize the tail energy by delaying the offloading requests, the dynamic TCP acknowledgment problem delays the acknowledgments to alleviate the acknowledgment overhead. That said, the two problems are actually quite different. In our problem, the tail energy is determined by the amount of time that the mobile device stays at the high-power state, since the transmission time is negligible comparing to the tail time; while in the dynamic TCP acknowledgment problem, the acknowledgment overhead mainly depends on the number of acknowledgements.

III. Coalesced Offloading: an Offline Solution

In this section, we start to solve the optimization problem that we have formulated by first transforming it into a discrete-time optimization problem, and then present an offline solution based on dynamic programming in this section.

A. From Continuous-Time to Discrete-Time Formulation

By carefully analyzing problem (1), we make the following two important observations.

Observation 2: All offloading requests should be transmitted either at their respective arrival times or the arrival time of other requests to minimize f_{cost}.

Proof: We prove that any transmission of requests that do not satisfy the above conditions increases the total costs. Suppose two requests arrive sequentially at time $\langle a_1, a_2 \rangle$, and we are to about to schedule their transmission time(s) g with the objective of minimizing f_{cost}. We have the following three choices: (1) $g \in (0, a_1]$, (2) $g \in (a_1, a_2)$, 3) $g \in [a_2, \inf)$.

For the first case, the total cost is

$$f_{\text{cost1}} = (a_1 - g) + \min\{a_2 - a_1, T\} + T.$$

Obviously, when $g = a_1$, the value is the minimum:

$$f_{\text{mincost1}} = \min\{a_2 - a_1, T\} + T.$$

In the second case, there would be an $\alpha(t-a_1)$ delay cost for the first request. Adding the same energy cost as in the first case, the total cost would take the form of

$$f_{\text{cost2}} = \min\{a_2 - t, T\} + T + \alpha(t - a_1),\ a_1 < t < a_2.$$

Similarly, the cost in the third case is

$$f_{\text{cost3}} = T + \alpha(a_2 - a_1) + 2\alpha(g - a_2),$$

of which the minimum value is:

$$f_{\text{mincost3}} = T + \alpha(a_2 - a_1)$$

when $g = a_2$.

From f_{cost2}, we have

- When $a_2 - t < T$,

$$f_{\text{cost2}} = \alpha(a_2 - a_1) + (1 - \alpha)(a_2 - t) + T.$$

Obviously, if $\alpha < 1$, $f_{\text{cost2}} > f_{\text{mincost3}}$. If $\alpha > 1$, then

$$\begin{aligned} f_{\text{cost2}} &> (t - a_1) + (a_2 - t) + T \\ &= a_2 - a_1 + T \geq f_{\text{mincost1}}. \end{aligned}$$

- When $a_2 - t > T$,

$$f_{\text{cost2}} = \alpha(t - a_1) + 2T > 2T \geq f_{\text{mincost1}}.$$

Therefore, no offloading request should be granted in times other than the arrival times if the total cost f_{cost} is to be minimized. All requests are either granted at their own arrival times or the arrival time of other requests.

■

Since t_j indicates the time when the wireless interface enters the high-power state, after which requests will be granted for transmission as they arrive, t_j equals to the arrival time of one of the requests. Similarly, s_j shall be a tail time T after some arrival time of a granted request. The original problem is equivalent to determining at which request's arrival the interface should be switched to the high-power state, and from which request's arrival no further transmissions should take place. For each request, the scheduling decision becomes whether to transmit it immediately or to wait until the next transmission. In this way, we can transform the original problem of deciding when to power on and off the network interface into making a decision for each request upon its arrival, on whether to send it out immediately or to delay it. Therefore, we may use 1 to represent transmitting the current request (with or without previously delayed requests), and use 0 to represent the decision to delay the current request. We transform the original problem (1) into deciding a *binary transmission sequence* $\langle 1, 0, 0, \ldots, 1, \ldots\rangle$ for the successively arriving requests, such that the total cost f_{cost} is minimized.

In a nutshell, if a request is granted immediately, the latency cost is 0, and the energy cost for transmitting the request arrives at a_i is $\min\{a_i - g_{\text{prev}}, T\}$, where g_{prev} represents the preceeding transmission; if the request is delayed, it only incurs a latency cost since it will not extend the tail time. Whenever the request is withheld from transmitting immediately, the latency cost is $\alpha(g_{\text{next}} - a_i)$. Thus, we have

$$f^i_{\text{cost}} = \begin{cases} \min\{a_i - g_{\text{prev}}, T\}, & \text{if granted,} \\ \alpha(g_{\text{next}} - a_i), & \text{if delayed.} \end{cases} \tag{2}$$

Let f_{cost} represent the sum of the energy cost and latency cost of transmitting the entire request sequence. We should

$$\min f_{\text{cost}} = \sum_{i=1}^{n} f^i_{\text{cost}}, \tag{3}$$

for 2^n combinations of binary transmission sequences according to Eqn. (2).

The transformation from the resulted binary transmission sequence into the time sets $\langle t_1, t_2, \ldots, t_k\rangle$ and $\langle s_1, s_2, \ldots, s_k\rangle$ is simple: let t_1 be the arrival time of the first 1 appearing in the sequence. Whenever the interface enters high-power state, a timer is set to T. If a request is granted before the timer counts down to 0, the timer will be reset to T. s_1 is the time that the timer firstly counts down to 0. Whenever the binary transmission sequence turns to 1 again, we set the arrival time of the request as t_2. In this way, we alternatively determine the time sequence of entering the high-power state $t_1, t_2, \ldots$ and $s_1, s_2, \ldots$ the time sequence of leaving that state.

B. Optimal Offline Algorithm

We now present an optimal *offline* algorithm to solve the problem (1), in which the arrival time sequence $a_1, a_2, \ldots, a_n$ are given *a priori*. The objective is to output a binary transmission sequence Seq$[n]$, such that the total cost f_{cost} is minimized. Though it depends on unrealistic assumptions of knowing the timing of all future requests, our offline algorithm will serve as the benchmark for us to design and evaluate our online algorithms.

We use dynamic programming to obtain an optimal offline algorithm with a time complexity of $O(n^2)$. Let $C_{\text{min}}[i]$ be the minimum cost of the arrival time subsequence $\langle a_1, a_2, \ldots, a_i\rangle$ and Seq$[i]$ be the binary transmission sequence for that arrival time subsequence. For an arrival time sequence of length i, there are 2^i possible combinations of binary transmission sequences, all of which will be traversed to obtain the one with the minimum cost.

With respect to the binary transmission sequence, we state the following facts that will lead us to the offline algorithm. There are 2^{i-1} possible binary transmission sequences in total for the arrival time sequence $\langle a_1, a_2, \ldots\ldots, a_i\rangle$, if the last request in the arrival time sequence must be transmitted. The cost of the arrival time sequence $\langle a_1, a_2, \ldots, a_i\rangle$ consists of the sum of $\min\{a_i - a_{i-1}, T\}$ and the cost for $\langle a_1, a_2, \ldots, a_{i-1}\rangle$, if the granting time sequence of the latter is a subsequence of the former. If not, the cost of the arrival time sequence from a_1 to a_i is the sum of the cost of the sequence from a_1 to a_{i-j}, the latency costs of the requests from a_{i-j+1} to a_{i-1}, and the tail time T. We first need to find out the granting time sequence that minimizes the cost of all the subsequences of $\langle a_1, a_2, \ldots, a_i\rangle$ to obtain the granting time sequence minimizing its total cost. We now give our optimal offline algorithm, as summarized in Algorithm 1.

Theorem 1: The offline algorithm produces an optimal solution to the transformed coalesced offloading problem (6).

Algorithm 1 The Offline Algorithm

```
Input: a_1, a_2, ..., a_n
Output: Seq[n]
Initialize C_min[0] = 0
Initialize Seq[0] = ⟨⟩
Initialize C_min[1] = T
Initialize Seq[1] = ⟨1⟩
for i ∈ [2, n] do
  C_min[i] = C_min[i − 1] + min{a_i − a_{i−1}, T}
  Seq[i] = ⟨Seq[i − 1], 1⟩
  for j ∈ [2, i] do
    C[i] = C_min[i − j] + α Σ_{k=i−j+1}^{i−1} (a_i − a_k) + T
    if C[i] < C_min[i] then
      C_min[i] = C[i]
      Seq[i] = ⟨Seq[i − j], 0_{1st}, ..., 0_{j−1th}, 1⟩
    end if
  end for
end for
```

Proof: It is known that if a problem possesses the optimal substructure property, then any dynamic programming algorithm that explores all subproblems is an optimal algorithm. To see problem (6) contains the optimal substructure property, we only need to note that if the sequence Seq$[i]$ optimizes the total cost of input $\langle a_1, a_2, \ldots, a_i\rangle$, the subsequence of Seq$[i]$ — Seq$[i-j]$ must be an optimal solution to the subsequence $\langle a_1, a_2, \ldots, a_{i-j}\rangle$. Our algorithm obviously explores all the possible subproblems to obtain the optimal solution. ■

IV. Ready, Set, Go: Online Algorithms

We are now ready to design *Ready, Set, Go* (RSG), our online algorithms to solve problem (1) without *a priori* knowledge of the arrival time sequence. We begin by considering the algorithms that probabilistically vary the amount of latency with a similar approach to the dynamic TCP acknowledgment problem [6]. We show that the coalesced offloading problem is a generalized case of this problem, which is known to be a generalization of the online ski rental problem.

A. The Dynamic TCP Acknowledgment Problem

The dynamic TCP acknowledgment problem is a generalization of the online ski rental problem with the following form. The input is a sequence of the packet arrival times $a_1, a_2, \ldots, a_n$ and the output is a set of times $t_1, t_2, \ldots, t_k$ at which an acknowledgment occurs. The latency is defined as the amount of time elapsed between a packet arrives and it is acknowledged. The cost of each acknowledgment is 1. The problem objective is to minimize

$$k + \sum_{1\le j\le k} \text{latency}(j).$$

It has been proved by Karlin *et al.* that the randomized algorithm of this problem has an optimal competitive ratio of $e/(e-1)$.

We can see that the coalesced offloading problem is a generalized case for the dynamic TCP acknowledgment problem with the following analysis.

While the cost for each acknowledgment in the dynamic TCP acknowledgment problem is a constant, its counterpart in our problem, *i.e.,* the energy consumption of each transmission, is a function that depends on the previous transmission time. To show the dynamic TCP acknowledgment problem is a special case of our problem, we only need to set T such that $T < (g_i - g_{i-1})$. Then the energy cost for each transmission is a constant T. If we set both T and α to 1, then the energy cost is exactly the acknowledgment cost of the TCP problem, whereas the latency costs in both problems are equivalent.

B. The Online Algorithm A_θ

Our algorithm A_θ is defined as follows.

Definition 1: A_θ is a randomized algorithm that selects θ between 0 and 1 according to a probability density function $p(\theta) = e^\theta/(e-1)$. Let $R(t, t')$ be the number of requests that arrive between time t and t', and $g_1, g_2, \ldots, g_i, \ldots$ be the times at which requests are granted and transmitted. Algorithm A_θ grants the next request at g_{i+1} such that there exists a time τ_{i+1}, $g_i < \tau_{i+1} < g_{i+1}$, that satisfies

$$R(g_i, \tau_{i+1})(g_{i+1} - \tau_{i+1}) = (\theta/\alpha)S_i, \tag{4}$$

where

$$S_i = 2(\min\{\tau_{i+1}-g_i, T\}+\min\{g_{i+1}-\tau_{i+1}, T\})-\min\{g_{i+1}-g_i, T\}.$$

The intuition behind the equation above is simple: given the previous transmission occurring at time g_i, the additional transmission happens at τ_{i+1} will reduce the latency cost by θS_i. S_i is essentially the amount of energy cost increment due to the additional transmission. It is easy to prove that

$$\min\{\tau_{i+1}-g_i, T\}+\min\{g_{i+1}-\tau_{i+1}, T\} \ge \min\{g_{i+1}-g_i, T\},$$

so that

$$S_i \ge \min\{g_{i+1} - g_i, T\} \ge \min\{g_{i+1} - \tau_{i+1}, T\}. \tag{5}$$

Fig. 3 helps to explain our algorithms and proofs. The x-axis represents the time, and the y-axis represents the number of request arrivals. The staircase function indicates the arrival sequence of the requests. g_i defines the times at which a bundle of requests is granted. The shaded area below the staircase curve and the dotted line above represents the saved latency cost. Fig. 3 shows an example of the online algorithm A_1, letting $\tau_0 = g_0 = 0$.

C. Deterministic Online Algorithm: Performance Analysis

We prove that when $\theta = 1$, the deterministic online algorithm A_1 is 2-competitive against the optimal algorithm A_{OPT}.

Lemma 1: The optimal algorithm grants a request between any pair of successive transmissions.

Proof: We suppose that A_1 grants requests at times $g_1, g_2, \ldots, g_i, \ldots$. Enrich the sequence by adding transmissions at time τ_{i+1}, $g_i < \tau_{i+1} < g_{i+1}$ for all i such that Eqn. (4) is satisfied. It is obvious to see from Fig. 4 that, by adding a transmission at time τ_{i+1}, the latency cost decreases at least by $\min\{(g_{i+1} - \tau_{i+1}), T\}$ units between g_i and g_{i+1},

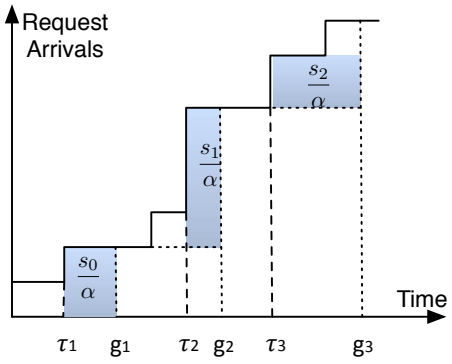


Fig. 3. The online algorithm A_1.

whereas the additional energy consumption incurred is at most $\min\{(g_{i+1} - \tau_{i+1}), T\}$ units. In this case, the new sequence is at least as good as the original one. It is easy to see the reduced amount of latency cost by Eqn. (5). To see the additional energy cost incurred is at most $\min\{(g_{i+1} - \tau_{i+1}), T\}$, we recall that by performing an additional transmission at τ_{i+1}, the energy cost is increased by $\min\{\tau_{i+1} - g_i, T\}$ whereas the original energy cost of the transmission at g_{i+1} $\min\{g_{i+1} - g_i, T\}$ is now updated to $\min\{g_{i+1} - \tau_{i+1}, T\}$ for one additional transmission. The net increase of the energy cost satisfies

$$\begin{aligned}\min\{(\tau_{i+1} - g_i), T\} - \min\{g_{i+1} - g_i, T\} \\ + \min\{g_{i+1} - \tau_{i+1}, T\} \\ \leq \min\{(g_{i+1} - \tau_{i+1}), T\}\end{aligned}$$

since $\tau_{i+1} < g_{i+1}$. Hence, there exists an optimal sequence that grants requests as least once in the interval (g_i, g_{i+1}) for all i. ■

Theorem 2: Algorithm A_1 is 2-competitive.

We leave the proof of the competitive ratio for the deterministic online algorithm A_1 in our technical report [7].

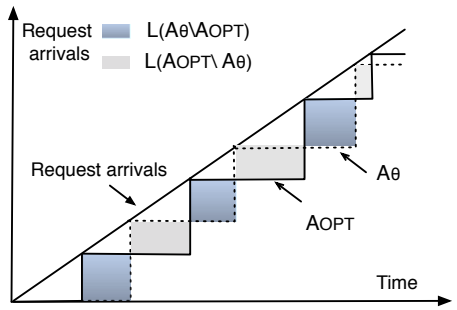


Fig. 4. The proof of the competitive ratio of A_θ.

D. Performance Analysis of A_θ

Theorem 3: The competitive ratio between the expected cost incurred by A_θ and the optimal cost is $e/(e-1)$.

Proof: We start our proof by first decomposing the total cost of A_θ. As Fig. 4 has illustrated, $L(A_\theta \backslash A_{\text{OPT}})$ is the latency incurred by A_θ but not A_{OPT}, which is the dark shaded area above the dotted line and below the solid line. Likewise, $L_(A_{\text{OPT}} \backslash A_\theta)$ stands for the latency incurred by A_{OPT} but not A_θ, which is illustrated by the light shaded area above the solid line and below the lighted line. The latency cost of A_θ is the area above the curve of A_θ and below the curve of request arrivals, which is at most the area above the solid curve plus the dark shaded area minus the light shaded area. Thus, the total cost satisfies

$$\begin{aligned}C_{A_\theta} \leq E_\theta + (C_{\text{OPT}} - E_{\text{OPT}}) \\ + [L(A_\theta \backslash A_{\text{OPT}}) - L(A_{\text{OPT}} \backslash A_\theta)] \times \alpha,\end{aligned}$$

letting E_θ and E_{OPT} be the energy cost of A_θ and A_{OPT}, respectively. By the definition of A_θ, the sum of the dark shaded area is:

$$\begin{aligned}L(A_\theta \backslash A_{\text{OPT}}) \leq (\theta/\alpha) \sum_i S_i \\ = (\theta/\alpha)(2E_{\text{OPT}} - E_\theta).\end{aligned}$$

For the light shaded area, we will prove the following lemma.

Lemma 2: The light shaded area $L(A_{\text{OPT}} \backslash A_\theta)$ satisfies:

$$\alpha L(A_{\text{OPT}} \backslash A_\theta) \geq \int_0^1 E(x)dx - (1-2\theta)E_{\text{OPT}} - \theta E_\theta. \quad (6)$$

To prove the lemma above, we make the following claim. Let $M(E, \theta)$ be the minimum, over all possible granting sequences W with the energy cost E, of the area above W and below the A_θ curve as shown in Fig. 5. We omit the request arrivals and use a line to represent A_θ. We claim that for any $u > v \geq \theta$,

$$M(E_u, \theta) \geq [(v-\theta)/\alpha](E_v - E_u) + M(E_v, \theta) \quad (7)$$

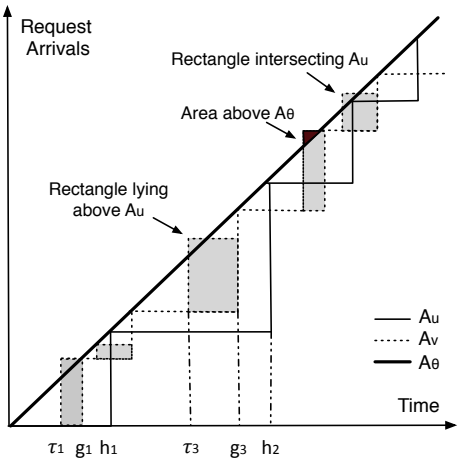


Fig. 5. The Proof of Lemma 3 (to prove the competitive ratio of A_θ)

Proof: Let n_u and n_v represent the total number of grants incurred by performing algorithms A_u and A_v for the same input. The granting sequence is $h_1, h_2, \ldots, h_{n_u}$ for A_u and is $g_1, g_2, \ldots, g_{n_v}$ for A_v. As shown in Fig. 5, the shaded rectangles of A_v, defined by the definition of A_v, intersect with the A_u curve at most n_u times. Therefore, at least $n_v - n_u$ shaded rectangles strictly lie above the curve of A_u. Pick

exactly $n_v - n_u$ of them, denote each one of them by its transmission sequence number i, and define the set of these rectangles as V^*. Let

$$S(V^*) = \sum_{i \in V^*} S_i.$$

Then the sum of the area of the $n_v - n_u$ rectangles in V^* is $(v/\alpha)S(V^*)$, and the area of $(v/\alpha)S(V^*)$ that lies above the curve of A_θ is at most $(\theta/\alpha)S(V^*)$. Thus the shaded area below the A_θ curve is at least $\frac{(v-\theta)}{\alpha}S(V^*)$, and this area strictly lie above the curve of A_u. We next generate a new granting sequence $g_1^*, g_2^*, \ldots, g_n^*$ with A_v^* such that the energy cost of it is exactly the same with the energy cost of the transmission sequence of A_v. Also, the new generated sequence issues a grant at $\tau_i, \forall i \in V^*$. Thus, the shaded area strictly lies below the curve of A_v^* but above the curve of A_u in Fig. 5, which is

$$M(E_u, \theta) - M(E_v, \theta) \geq [(v - \theta)/\alpha]S(V^*). \quad (8)$$

Note that E_v in Eqn. (8) is the energy cost of the new granting sequence of A_v^*, which is equivalent to the energy cost of A_v. It is still required to prove the following to have Eqn. (7).

$$S(V^*) \geq (E_v - E_u). \quad (9)$$

By Eqn. (5), we have verified

$$S(V^*) \geq \sum_{i \in V^*} \min\{g_{i+1} - g_i, T\}.$$

For the same input sequence, the entire time duration of the transmission sequences of A_u and A_v is the same:

$$\sum_{i=0}^{n_u-1} (h_{i+1} - h_i) = \sum_{i=0}^{n_v-1} (g_{i+1} - g_i),$$

Removing the $n_v - n_u$ items in V^* from the right-hand side and applying the minimum function to both sides, we then get

$$\sum_{i=0}^{n_u-1} \min\{h_{i+1} - h_i, T\} \geq \sum_{j \notin V^*} \min\{g_{j+1} - g_j, T\},$$

Therefore,

$$\begin{aligned} S(V^*) &\geq \sum_{i=0}^{n_v-1} \min\{g_{i+1} - g_i, T\} - \sum_{j \notin V^*} \min\{g_{j+1} - g_j, T\} \\ &\geq \sum_{i=0}^{n_v-1} \min\{g_{i+1} - g_i, T\} - \sum_{i=0}^{n_u-1} \min\{h_{i+1} - h_i, T\} \\ &= (E_v - E_u). \end{aligned}$$

Combining with Eqn. (8) gives us Eqn. (7). ■

By rewriting and integrating Eqn. (7), we prove Lemma 3, the detail of which is given in our technical report [7]. Also, the rest of the proof of Theorem 2 is given in [7] due to space constraints. ■

V. Performance Evaluation

We evaluate both offline and online algorithms using both model-driven simulations and real-world experiments on a mobile device. We start with measuring the tail time in our model to evaluate the cost performance of both offline and online algorithms. Then we quantify the reduction of energy utilizations performing the RSG algorithms with real-world runtime traces from mobile applications.

We run all of our real-world experiments on an iPhone 3GS with iOS 6.1.3, and using the Bell Mobility 3G cellular network. To measure the energy consumption, we use PowerGremlin [8], a power usage monitor application, to record run-time battery capacity (mAh) with a sample duration of one second. All of our measurements are performed under stable network conditions, with the mobile device running in a standalone environment in which all other applications and background tasks are shut off except for our application-level prototype service, and with the screen off.

A. Measuring the Tail Time

The measurement methodology of the tail time is as follows. Initially we plan to use PowerGremlin to track the energy trace of sending a packet, however, this method ends up with a very subtle energy change that is difficult to detect. We decided that the tail time is to be measured by transmitting successive packets of equal sizes, given that the time intervals between every two transmissions are the same. Our argument is that when the time interval is smaller than tail time T, the 3G network interface is kept on from the previous transmission to the next, thus the variation of the transmission interval will make no difference to the overall energy consumption. On the contrary, if the time interval between transmissions is longer than the tail time T, the 3G network interface enters stand-by some time T after the completion of the last transmission, thus the overall energy consumption is reduced.

To measure the 3G tail time, we generate stable sequential offloading requests over a period of 5 minutes. To eliminate the effect of varying transmission costs incurred by different sizes of the packets, we set the packet to be of equal sizes, and small enough to avoid a heavy transmission overhead. In our experiments, the time intervals between requests are designated to span from 3 to 17 seconds. Our measurement result is in accordance with our argument: the total energy consumed during the 5-min period keeps the same level when the transmission time interval varies from 3 to 9 seconds, but drops dramatically at 9 seconds. As a result, 9 seconds is the tail time for the 3G interface in iOS 6. We hereby use the result in our subsequent simulations and experiments.

B. Model-Driven Evaluation

In our model-driven evaluations, the trace of offloading requests is a sequence of the arrival times $\langle a_1, a_2, \ldots, a_i, \ldots \rangle$, simulating the timing of multiple offloading requests from several simultaneously running applications. We categorize the request patterns into three types: low, medium, and high fluctuation. With these request sequences as input, we compare

TABLE I
ENERGY COST REDUCTION COMPARED WITH THE NAIVE STRATEGY.

α	Offline	Randomized	Deterministic
0.3	62.3%	28.2%	14.84%
1.0	36.23%	14.33%	8.49%
1.3	34.09 %	13.86%	8.61%

the total cost f_{cost} of the online RSG algorithms with the benchmark of the offline algorithm. Each simulation result is averaged over 500 rounds of tests.

As Fig. 6(a) shows, on average, the cost of the randomized online RSG is no more than 1.4009 times of the benchmark offline algorithm, while the ratio of the cost of the deterministic online RSG to the offline algorithm is 1.4652. Both numbers are within the 1.58 and 2-competitive ratio as analyzed previously. In addition, the randomized online algorithm generally achieves better performance in terms of f_{cost} when the fluctuation is higher, while the performance of the deterministic online algorithm almost remains the same for different inputs.

Fig. 6(b) compares the energy costs of the naive, deterministic online, randomized online, and offline algorithms when the weight factor α varies. The naive strategy is to send the offloading request upon its arrival. As stated previously, the energy cost is proportional to the time that the network interface stays in the high-power state, which can be used to estimate the energy costs. As Fig. 6(b) illustrates, the naive strategy incurs the highest energy costs over all the αs. The performance of the offline algorithm dominates when α is less than 1, but approaches the curve of the naive case when α increases. This conforms with our observation that the offline algorithm grants more requests immediately upon arrival when more weights are added to the latency. The curves for the two RSG online algorithms lie between the naive and the offline ones, and their energy cost curves climb less dramatically than the offline curve.

C. Experiments on the Mobile Phone

In our real-world experiments, we choose XML-RPC [9] to emulate successive offloading requests generated from multiple applications and their transfers to the cloud. We use three typical types of offloading requests — random, bursty, and stable — to represent real-world traces. Each measurement result is averaged over 50 trials, with each trial containing around 50 transfer requests. Without loss of generality, the parameter α is set to be 0.3. From our experimental results, as shown in Fig. 6(c), we have observed that the RSG deterministic online, randomized online, and offline algorithms can respectively achieve 20.23%, 27.10% and 60.20% of energy reduction on average, for all three types of requests, compared to the naive strategy.

When we look into that how the energy costs vary with α, we find that when α is smaller, RSG bundles requests in a more aggressive manner so that more energy is saved. Fig. 6(d) and Table I together illustrate the energy cost reduction of different algorithms compared to the naive strategy with varying α, with random requests only.

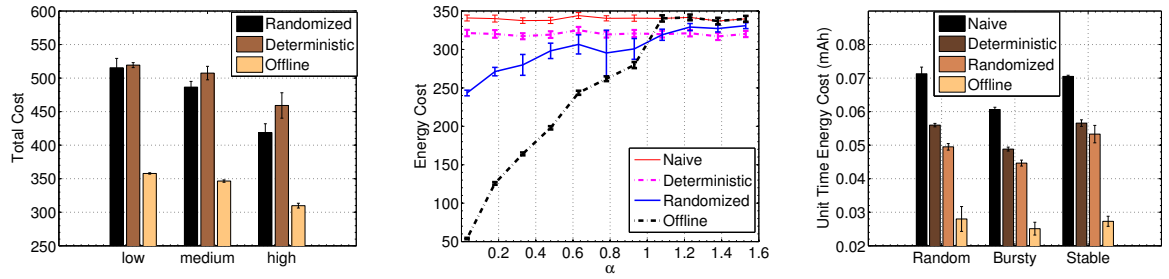
We take a step further to test our algorithms using real-world traces. To get the trace, we run three typical mobile applications, Rubik Solver, Email, and online chatting, that are ready to offload on our iPhone 3GS, and leverage Wireshark [10] to record their network traffic. The Rubik Solver demonstrates highly bursty traffic for it is computation-intensive, whereas Email regularly checks with the server in the background, and online chatting arbitrarily generates traffic from time to time. Fig. 6(e) shows the actual transmission times before and after scheduling. Apparently, with the RSG algorithm, requests from multiple applications are transmitted in bundles. To further verify our results, we monitor the raw battery voltage variation on the mobile device. As Fig. 6(f) shows, the battery voltage are more stable and decrease more moderately with RSG. Our experiments have revealed that by performing the RSG algorithm with our real-world traces, the energy consumption is reduced by 20.71%.

VI. RELATED WORK

Many existing works in the literature of code offloading between mobile devices and the cloud only considered the optimal offloading choice of a single application. Works such as [1], [3] decided at runtime which parts of the application are to be remotely executed with an optimization engine, in order to achieve the best energy savings. Kosta *et al.* [11] developed a framework of smartphone virtualization in the cloud, allowing method-level computation offloading. Gordon *et al.* [2] used a distributed shared memory technique instead of remote procedure calls to support multi-threaded applications to run on multiple machines. However, it has been observed that the on-and-off switching state of the network interface, incurred by offloading requests from multiple simultaneously running applications, unnecessarily consumes much idle energy. Without considering that aspect in the entire optimization framework, it is insufficient to discuss code offloading alone.

Our work is also closely related to Balasubramanian *et al.* [5], as it found that 3G incurs a high tail energy overhead for lingering in the high-power state after the completion of a transfer. It also proposed a scheduling algorithm to minimize the energy consumed while meeting user-specified deadlines. However, the scheme is only designed for delay-tolerant and prefetching applications, without taking the length of delays into account.

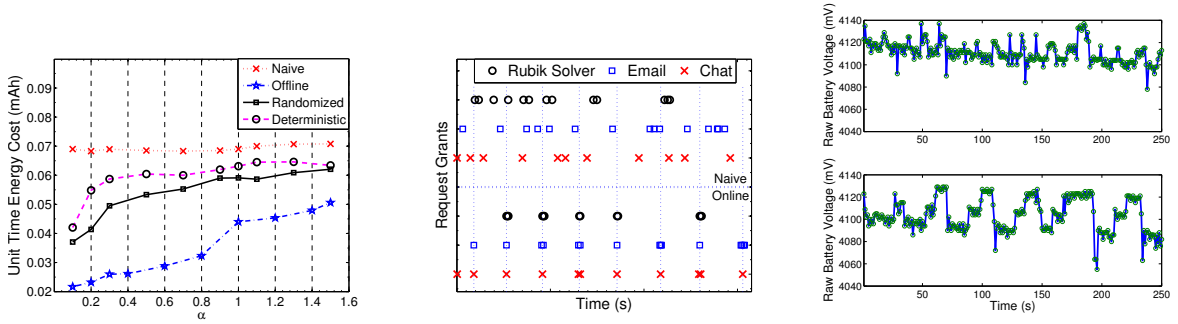
Our online strategies are tied to the online algorithm literature [6], [12], [13]. The dynamic TCP acknowledgment problem is a generalization of the classical ski rental problem with the same competitive ratio. We show that our problem is a generalization of the dynamic TCP acknowledgment problem, and we prove that the competitive ratio achieves its special case, which is already proven to be the best possible. A similar case can also be found in scheduling tasks to minimize the total power consumption [14], and they presented an effort to minimize the number of "gaps," *e.g.*, the idle periods, in application execution.



(a) The f_{cost} of offloading requests with different levels of fluctuations.

(b) The estimated energy cost with varying α.

(c) The energy consumption on the mobile device with different types of offloading requests.



(d) The energy consumption on the mobile device with a varying α.

(e) The request transmissions on the mobile device w/o the RSG algorithm.

(f) The battery voltage change as measured on the mobile device. The top figure shows the result of the naive strategy, and the bottom one shows the result of the RSG algorithm.

Fig. 6. RSG: Simulation and Experimental Results.

VII. Concluding Remarks

Coordinating the offloading requests of multiple applications to achieve greater energy savings while maintaining satisfactory performance is an important issue in offloading from mobile devices to the cloud. In particular, how can we schedule the offloading requests without any knowledge of the future requests? To answer that, we propose RSG, which consists of two online algorithms, one deterministic and one randomized, that dynamically decide when to grant requests without future information. We prove that the RSG online algorithm achieves the best possible 2-competitive ratio for the deterministic case and $e/(e-1)$ for the randomized one. With RSG, our real-world implementation on the iOS platform has shown a substantial amount of energy savings.

References

[1] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic Execution between Mobile Device and Cloud," *in Proc. 6th Conf. on Computer Systems*, 2011.

[2] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "COMET: Code Offload by Migrating Execution Transparently," *in Proc. 10th USENIX Conf. on OSDI*, 2012.

[3] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," *in Proc. 8th MobiSys*, 2010.

[4] B.-G. Chun and P. Maniatis, "Augmented Smartphone Applications Through Clone Cloud Execution," *USENIX HotOS Workshop*, 2009.

[5] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy Consumption in Mobile Phones: a Measurement Study and Implications for Network Applications," *in Proc. 9th ACM SIGCOMM Conf. on IMC*, 2009.

[6] A. R. Karlin, C. Kenyon, and D. Randall, "Dynamic TCP Acknowledgment and Other Stories about e/(e-1)," *in Proc. 33rd ACM Symposium on Theory of Computing*, 2001.

[7] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li, "Ready, Set, Go: Coalesced Offloading from Mobile Devices to the Cloud," 2013. [Online]. Available: http://iqua.ece.toronto.edu/~bli/papers/rsg.pdf

[8] "Powergremlin." [Online]. Available: https://github.com/palominolabs/powergremlin

[9] "Cocoa xml-rpc framework." [Online]. Available: https://github.com/corristo/xmlrpc

[10] "Wireshark." [Online]. Available: http://www.wireshark.org/

[11] 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 Proc. IEEE INFOCOM*, 2012.

[12] D. R. Dooly, S. A. Goldman, and S. D. Scott, "TCP Dynamic Acknowledgment Delay: Theory and Practice," *in Proc. 30th ACM Symposium on Theory of Computing*, 1998.

[13] W. Wang, B. Li, and B. Liang, "To Reserve or Not to Reserve: Optimal Online Multi-Instance Acquisition in IaaS Clouds," *in Proc. IEEE/ACM ICAC*, 2013.

[14] P. Baptiste, "Scheduling Unit Tasks to Minimize the Number of Idle Periods: a Polynomial Time Algorithm for Offline Dynamic Power Management," *in Proc. 17th ACM-SIAM Symposium on Discrete Algorithm*, 2006.