Application-Aware Traffic Scheduling for Workload Offloading in Mobile Clouds

Liang Tong and Wei Gao
Department of Electrical Engineering and Computer Science
University of Tennessee at Knoxville
ltong@vols.utk.edu, weigao@utk.edu

***Abstract*—Mobile Cloud Computing (MCC) bridges the gap between limited capabilities of mobile devices and the increasing complexity of mobile applications, by offloading the computational workloads from local devices to the cloud. Current research supports workload offloading through appropriate application partitioning and remote method execution, but generally ignores the impact of wireless network characteristics on such offloading. Wireless data transmissions incurred by remote method execution consume a large amount of additional energy during transmission intervals when the network interface stays in the high-power state, and deferring these transmissions increases the response delay of mobile applications. In this paper, we adaptively balance the tradeoff between energy efficiency and responsiveness of mobile applications by developing application-aware wireless transmission scheduling algorithms. We take both causality and run-time dynamics of application method executions into account when deferring wireless transmissions, so as to minimize the wireless energy cost and satisfy the application delay constraint with respect to the practical system contexts. Systematic evaluations show that our scheme significantly improves the energy efficiency of workload offloading over realistic smartphone applications.**

I. Introduction

Modern mobile devices continuously generate and process sensory data about the surrounding environment, and hence enable users' contextual awareness [12], [6]. Due to the computational intensity of processing such sensory data which could be audio, images, or biometric measurements, mobile devices tend to be operated in a mobile cloud computing (MCC) environment, in which the local computation workloads are offloaded to the remote cloud via cellular networks [14]. The involvement of mobile clouds hence augments local devices' capabilities and prolongs their battery lifetime.

The major challenge of MCC is that data transmission between mobile devices and the remote cloud through cellular networks is expensive. Hence, instead of unconsciously offloading the entire application processes for remote execution, researchers suggest to perform workload offloading at the fine-grained level of different application methods [13], [8], and supports such offloading via code migration [2] and Virtual Machine (VM) synthesis [1], [5]. A user application is adaptively partitioned according to the computational complexity and program state sizes of its methods, to ensure that the amount of local computation energy saved by workload offloading exceeds the expense of wirelessly transmitting the relevant program states to the remote cloud.

However, limited research has been done on investigating the characteristics of wireless networks during such offloading. These characteristics, however, could seriously degrade the energy efficiency of workload offloading if not appropriately handled. Remote method executions create bursts of wireless traffic with short intervals, during which the radio interface stays in the high-power state and consumes a large amount of additional energy [10]. A common solution to reducing such additional cost is to defer wireless transmissions and send data together as bundles [20], [17], but may increase the response delay and degrade the application performance. Ignorance of such energy-delay tradeoff, hence, becomes the major factor degrading the performance and energy efficiency of MCC.

The key to efficient balancing the energy-delay tradeoff in MCC is to develop efficient algorithms which appropriately schedule the wireless data transmissions incurred by remote method executions, taking both the wireless energy cost and application delay constraint into consideration. Development of such algorithms is challenging due to the interdependency between wireless transmissions and mobile application executions. First, a modern mobile device can execute multiple applications simultaneously, and method executions belonging to the same application have causality with each other, such that a method cannot be executed until all the prior method executions have completed. This causality makes it difficult to ensure the global scheduling optimality for all transmissions while satisfying their interdependency. Second, executions of mobile applications could be highly dynamic at run-time due to the different input data, user operations, and system contexts. Such run-time dynamics lead to uncertainty of the wireless transmission pattern and hence complicates the quantitative design of transmission scheduling algorithms.

In this paper, we propose novel schemes to address the above challenges and adaptively balance the tradeoff between energy efficiency and responsiveness of mobile applications. Being different from existing work which regardlessly defers wireless transmissions into groups, our basic idea is a fundamental shift: we take both causality and run-time dynamics of application method executions into account when deferring wireless transmissions, so as to minimize the wireless energy cost while satisfying the specified application delay constraints. More specifically, we first develop efficient algorithms for offline transmission scheduling in MCC, based on a pre-known sequence of transmissions to be scheduled. These scheduling algorithms ensure global minimization of transmission energy cost, and also enable flexible balancing between the energy and delay aspects in MCC at run-time. Furthermore, we incorporate the run-time dynamics of mobile application executions into transmission scheduling by developing a stochastic framework, so as to extend the transmission scheduling algorithms to online operations by probabilistically predicting the application execution path in the future. To the

This work was supported in part by the National Science Foundation (NSF) under grants CNS-1456656 and CNS-1526769, and was supported by the Army Research Office (ARO) under grant W911NF-15-1-0221.

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

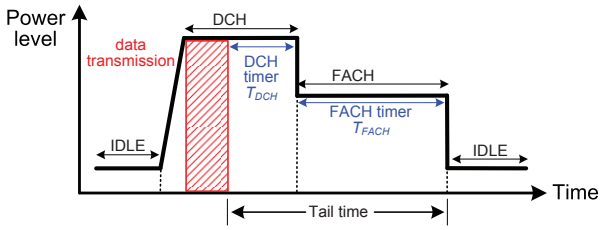


Fig. 1. Energy model of the UMTS cellular radio interface

best of our knowledge, our work is the first which explicitly exploits the characteristics of mobile application executions, especially the various constraints posed by these applications on wireless transmissions, for traffic scheduling in MCC.

The rest of this paper is organized as follows. Section II reviews the existing work. Sections III and IV describe motivation and the big picture of our approach. Sections V and VI present the details of the proposed transmission scheduling algorithms. Section VII evaluates the performance of our proposed approach. Section VIII concludes the paper.

II. Related Work

Workload offloading in mobile clouds focuses on the questions of *how* to offload and *what* to offload. Synthesis and migration of Virtual Machines (VMs) have been widely studied to improve the efficiency and reliability of remote code execution [1], [8]. Other schemes further improve the offloading generality by supporting multi-threaded [5] and interactive applications [12]. Appropriate decisions of application partitioning have also been studied based on the profiling data about application execution and system contexts, and obtain such profiling data via either benchmark testing [2], [1] or online application profilers [8], [17]. In this paper, we adopt the above research literature for decisions of application partitioning and operations of remote method execution.

The efficiency of MCC operations depends on wireless data transmissions. Existing schemes suggest to reduce the wireless energy cost by scheduling transmissions based on the signal strength [15], optimizing the network timer configurations [18], or exploiting the fast dormancy feature [11]. Grouping multiple transmissions as bundles has also been proposed for energy saving [20], [17]. However, the impact of high-level application executions on wireless network operations is generally ignored by these schemes. In contrast, our proposed transmission scheduling algorithms consider the run-time application executions as an integral part determining wireless network behaviors, and hence achieve much higher energy efficiency of wireless data transfer.

The wireless energy cost and application performance of workload offloading have also been considered in the same domain [17], which assumes that application method invocations are independent and homogeneous over time. Wireless transmissions from multiple applications are scheduled within a unified sequence without considering their mutual interdependency. The energy-delay tradeoff is then abstracted as a joint optimization problem without taking the run-time dynamics of application executions into account. Other schemes schedule the data traffic of mobile applications in different ways, either to optimize the application response delay [3] or reduce the energy consumption of wireless transmission [16], [9]. However, they only take the dynamic conditions of wireless link quality into account, without considering the long-tail characteristics of wireless links or the causality of data traffic incurred by application executions. Comparatively, in this paper we remove these assumptions from the transmission scheduling algorithm design, and hence ensure the practical applicability of the produced transmission schedules.

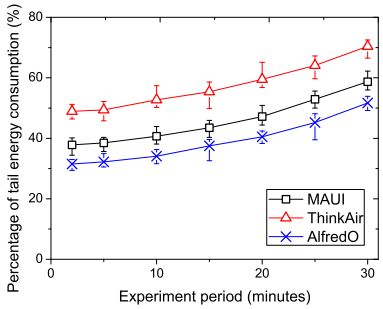


Fig. 2. Wireless energy consumption during the tail time

III. Background and Motivation

In this section, we motivate our design of application-aware transmission scheduling in MCC through experimental investigations of various aspects of MCC characteristics.

A. Energy Model of Cellular Networks

3G/4G cellular network is the most common way for mobile devices to interact with the cloud, when WiFi connectivity is unavailable. The energy model of a typical data transmission over UMTS-based 3G/4G cellular networks is shown in Figure 1. The cellular radio interface can work at three states: IDLE, DCH, and FACH. It stays in the IDLE state and consumes little power when being turned on. Whenever there is a transmission request, the radio allocates dedicated transport channels (DCH) and promotes to the DCH state. After transmission completes, instead of immediately resuming to the IDLE state, the radio demotes to the forward access channel (FACH) state which only allows low-speed transmission (< 15 kbps). The energy level of FACH is lower than that of DCH, but is still substantially higher than that of IDLE. Such state demotions are controlled by two inactivity timers, whose values are usually a few seconds and vary among different carriers [10].

B. Additional Energy Cost during Tail Times

The above energy model indicates that data transmissions caused by remote execution of application methods may incur a significant amount of additional energy cost during the "*tail time*", when the cellular radio interface stays in the high-power DCH and FACH states after the completion of data transmissions. To further verify such additional energy cost, we implemented various existing MCC schemes including including MAUI [2], ThinkAir [8], AlfredO [13] over a Samsung Galaxy S4 smartphone running Android v4.4.4, and use these existing schemes to offload the workloads of Firefox for Mobile v24.0 application[1] to a remote cloud server. The experiment results in Figure 2 were obtained by having four graduate students randomly browse 50 different webpages at their own choices, and show that these offloading schemes

[1]http://hg.mozilla.org/mozilla-central/archive/tip.tar.gz

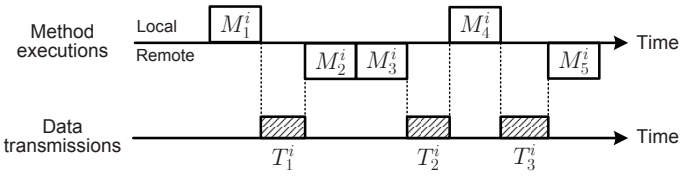


Fig. 3. System model of mobile cloud application executions

may spend up to 70% of the wireless energy during the tail time. Such energy cost, in particular, will significantly increase when the application is running for a longer period of time. These results suggest grouping data transmissions as bundles to avoid frequent state transitions and minimize the length of the tail time. The formation of such bundles, however, is affected by the application delay constraints.

IV. Overview

In this section, motivated by the above results, we present the high-level ideas of our proposed approach to application-aware wireless transmission scheduling in MCC, with respect to the notations in Table I being used throughout the paper.

TABLE I
Notation Summary

Notation	Explanation
D_i	The delay constraint of application i
n_i	The number of methods specified in application i
M_k^i	The k-th method of application i invoked in an execution path
T_j^i	The j-th wireless transmission for executing application i
t_s^j, t_e^j	The starting and ending times of T_j^i
d_j	The scheduled delay of transmission T_j^i
$d_{\max}$	The maximum amount of delay in order to avoid overlap among transmissions
j_P	The separation point between Stage 1 and Stage 2 of offline transmission scheduling
s_m	The m-th state in the order-k semi-Markov model formulating dynamic method transitions
$\tilde{s}_m$	The m-th composite Markovian state $\{s_{m-1}, ..., s_{m-k}\}$
$H_{ij}(t)$	Sojourn time distribution transiting from $i \in \mathbb{S}^k$ to $j \in \mathbb{S}$

A. System Model

We consider a generic scenario of MCC which is also used in [17] and consists of multiple applications running concurrently. Each application i has its own delay constraint D_i that can be specified offline or adaptively changed online. For an application i, we adopt the same model in [2] and denote its execution path in a specific run as $\{M_1^i, M_2^i, ..., M_{n_i}^i\}$, such that a method M_k^i is only invoked after the executions of all prior methods $\{M_j^i\}$ $(j < k)$ have completed. i's execution paths in different runs may be heterogeneous.

In this paper, we refer to the existing work to determine the set of application methods being offloaded [2], [8]. As shown in Figure 3, the application data only needs to be wirelessly transmitted between executions of M_k^i and M_{k+1}^i if these two methods are executed at different network locations in MCC, and the intervals between transmissions are hence constrained by the method execution times. Since the elapsed time for transmitting the program states for a mobile application is usually very short, we assume the wireless network condition to be stable during any single transmission lifespan. The specific elapsed time for each data transmission is then determined by the specific data size being transferred. Consideration of dynamic conditions of the wireless channel [19], on the other hand, is orthogonal to the major focus of this paper.

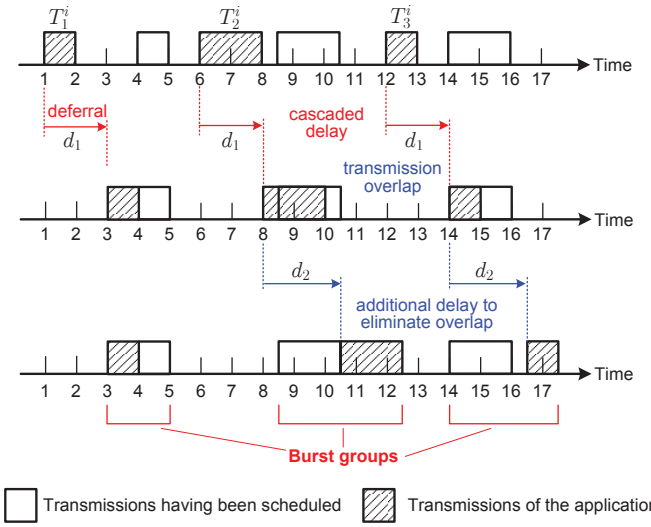


Fig. 4. Wireless transmission scheduling through transmission deferrals

B. The Big Picture

Our basic idea of minimizing the energy cost of wireless transmissions in MCC is to schedule these transmissions as a number of "burst groups" and minimize the intervals between transmissions in each burst group. As a result, we eliminate the additional wireless energy cost during the tail times while satisfying the delay constraints of mobile applications.

The big picture of our approach is illustrated in Figure 4, in which transmissions of an application i are scheduled with respect to transmissions of other running applications that have already been scheduled. Each burst group consists of a transmission T_j^i of application i and the next transmission having been scheduled in the future. Hence, we aim to defer T_j^i to minimize the transmission interval within each burst group. For example in Figure 4, T_1^i is delayed for $d_1 = 2$ seconds to save its tail energy. Other transmissions of application i could be deferred in the same way, until the application's delay constraint is reached. This scheduling strategy can be iteratively operated for each application, so as to support dynamic multi-threaded mobile applications.

Such separate deferrals of transmissions, however, would be difficult due to the causality among transmissions. For example, in Figure 4 when T_1^i is delayed for 2 seconds, this delay also applies to T_2^i due to the computation needed between T_1^i and T_2^i, and results in overlap between T_2^i and another scheduled transmission. To avoid contention on the wireless channel, T_2^i has to be further delayed for another $d_2 = 2.5$ seconds. This additional delay increases the response time of mobile applications. It will also be recursively applied to all subsequent transmissions of application i and complicates the transmission scheduling. In Section V, we will address the challenges raised by such cascaded delay and transmission overlap by developing offline scheduling algorithms. In Section VI, we will incorporate run-time dynamics of application executions and develop online scheduling algorithms.

V. Offline Transmission Scheduling

In this section, we propose offline transmission scheduling algorithms based on the complete knowledge about the sequence of transmissions $T_1^i, ..., T_{n_i}^i$ to be scheduled[2], whose

[2]For simplicity, we omit the subscript and superscript i in the relevant notations throughout the rest of this paper.

starting and end times are originally determined prior to execution. In practice, knowledge about these transmissions could be obtained through static analysis of the application binaries prior to the actual application execution, e.g., derivation of the application's calling graph [2]. Our scheduling algorithms aim to determine the amount of time (d_j) that each transmission T_j is deferred, with respect to the existing set of transmissions denoted as $\{P_1, P_2, ..., P_{n_s}\}$. These existing transmissions will remain unchanged during the scheduling process.

A. Problem Formulation

As shown in Figure 4, when a transmission T_j is delayed for d_j, it may overlap another existing transmission $P_{j'}$. Let $\tau_s^{j'}$ and $\tau_e^{j'}$ indicate the starting and ending time of $P_{j'}$, respectively, and $I_j(d_j)$ be an identifier indicating whether T_j overlaps with $P_{j'}$ after being deferred for d_j, we have

$$I_j(d_j) = \begin{cases} 1, & \text{if } t_s^j + d_j \in [\tau_s^{j'}, \tau_e^{j'}) \text{ or } t_e^j + d_j \in (\tau_s^{j'}, , \tau_e^{j'}] \\ 0, & \text{otherwise} \end{cases}$$

Similarly, let $R_j(d_j) \in \{0, 1\}$ indicate whether the two transmissions T_j and $P_{j'}$ are "bundled", i.e., the transmission interval between T_j and $P_{j'}$ is 0, we have $R_j(d_j) = 1$ if and only if $t_s^j + d_j = \tau_e^{j'}$ or $t_e^j + d_j = \tau_s^{j'}$. As a result, we determine d_j for each T_j, so as to maximize the total number of such bundles while satisfying both the causality among transmissions and the delay constraints of mobile applications. We formulate such scheduling problem as follows:

$$\begin{aligned} &\max \sum_{j=1}^{n_i} R_j(d_j) \\ &\texttt{s.t. } d_j \le d_{j+1}, I_j(d_j) = 0, \end{aligned} \tag{1}$$

Without loss of generality, we consider the value of each d_j is an integer in $[0, D_i]$. In practical mobile systems, d_j could be counted in either the number of milliseconds or CPU cycles, both of which can be precisely measured by current mobile devices. Then, the problem in (1) becomes an integer programming problem, and the two constraints represent transmission causality and elimination of overlap between transmissions, respectively. It is easy to see that the solution space of this problem is $D_i^{n_i}$, which requires an exponential time complexity to solve by exhaustive search. Instead, we develop various scheduling algorithms that can solve this problem in more computationally efficient ways.

B. Optimal Transmission Scheduling

First, we design an offline transmission scheduling algorithm which ensures global scheduling optimality via dynamic programming. The basic idea is to break the problem down to a collection of simpler subproblems. We solve different subproblems and then combine the solutions to all subproblems together to produce an overall solution.

Let $\texttt{BUNDLE}(n_i, D_i)$ denote the transmission scheduling problem described by Eq. (1). In order to exploit the dynamic programming approach for transmission scheduling, we first prove that $\texttt{BUNDLE}(n_i, D_i)$ has optimal substructure. A problem exhibits optimal substructure if an optimal solution to this problem contains optimal solutions to all of its subproblems.

Lemma 1: $\texttt{BUNDLE}(n_i, D_i)$ has optimal substructure.

Proof: Let $(d_1, d_2, \ldots, d_n)$ be an optimal solution to $\texttt{BUNDLE}(n, D)$, we prove that $(d_1, d_2, \ldots, d_{n-1})$ must be an optimal solution to $\texttt{BUNDLE}(n, d_n)$ via contradiction.

Algorithm 1: Optimal transmission scheduling

```
dp[0][k] ← 0, k ∈ {0, 1, ..., n_i}
for j = 1 to n_i do
    for k = 1 to D_i do
        if I_j(k) = 1 then
            dp[j][k] ← −∞
        end
        else
            if R_j(k) = 1 then
                dp[j][k] ← max_{l≤k} {dp[j − 1][l]} + 1
            end
            else
                dp[j][k] ← max_{l≤k} {dp[j − 1][l]}
            end
        end
    end
end
k ← D_i
for j = n_i to 1 do
    d_j ← min{arg max_{l≤k} dp[j][l]}
    k ← d_j
end
```

Since d_n is the delay for T_n, $R_n(d_n)$ can be 0 or 1. When $R_n(d_n) = 0$, this indicates that $\sum_{j=1}^{n} R_j(d_j) = \sum_{j=1}^{n-1} R_j(d_j)$. If $(d_1, d_2, \ldots, d_{n-1})$ is not an optimal solution for $\texttt{BUNDLE}(n, d_n)$, there must exist $(d_1^{'}, d_2^{'}, \ldots, d_{n-1}^{'})$ which make $\sum_{j=1}^{n-1} R_j^{d_j^{'}} > \sum_{j=1}^{n-1} R_j^{d_j}$. As a result, $\sum_{j=1}^{n-1} R_j^{d_j^{'}} + d_n > \sum_{j=1}^{n} R_j^{d_j}$, which contradicts with the fact that $(d_1, d_2, \ldots, d_n)$ is an optimal solution to $\texttt{BUNDLE}(n, D)$. Similarly proof can be applied when $R_n^{d_n} = 1$, and this lemma is proved by the existence of such contradiction. ■

Based on the optimal substructure, let $\texttt{dp}[j][k](j \le D)$ be the optimal number of bundles for a subproblem of transmissions $\{T_1, T_2, ..., T_j\}$ and T_j is delayed for k, we have

$$\texttt{dp}[j][k] = \begin{cases} -\infty, & I_j(k) = 1 \\ \max\limits_{l \le k}\{\texttt{dp}[j-1][l]\} + 1, & I_j(k) = 0, R_j(k) = 1 \\ \max\limits_{l \le k}\{\texttt{dp}[j-1][l]\}, & I_j(k) = 0, R_j(k) = 0 \end{cases}$$

The above equation shows how the value of $\texttt{dp}[j][k]$ can be recursively computed. When $I_j(k) = 1$, the solution $(d_1, d_2, \ldots, d_j)$ is not feasible because T_j overlaps with other transmissions. Considering that $\texttt{BUNDLE}(n_i, D_i)$ is a maximization problem, we make $\texttt{dp}[j][k] = -\infty$. Otherwise when $I_j(k) = 0$, $\texttt{dp}[j][k]$ can be derived from its subproblems corresponding to $\texttt{dp}[j-1][l](l \le k)$. Eq. (V-B), therefore, also shows that $\texttt{BUNDLE}(n_i, D_i)$ has overlapping subproblems.

Based on Eq. (V-B), we propose our optimal scheduling algorithm in Algorithm 1. We first compute all the optimal values $\texttt{dp}[j][k](j \le n_i, k \le D_i)$ and then use these values to derive the optimal solutions $d_j(j \le n_i)$. For each j, there may be multiple k which maximize $\texttt{dp}[j][k]$, and we always choose the minimum value of k to be d_j. The optimality of this algorithm is proved by the following theorem.

Theorem 1: Algorithm 1 produces an optimal solution to the transmission scheduling problem $\texttt{BUNDLE}(n_i, D_i)$.

Proof: We have proved that $\texttt{BUNDLE}(n_i, D_i)$ has optimal substructure and overlapping subproblems in Lemma 1 and Eq. (V-B), respectively. Algorithm 1 also indicates that it explores every subproblem of $\texttt{BUNDLE}(n_i, D_i)$. Theorem 1 is hence proved by combining these facts. ■

In Algorithm 1, there are $n_i \times D_i$ values for each $\texttt{dp}[j][k](j \leq n_i, k \leq D_i)$, and each $\texttt{dp}[j][k]$ is derived by comparing at most D_i subproblems. Therefore, the computational complexity of Algorithm 1 is $O(n_i D_i^2)$.

Algorithm 2: Greedy ransmission scheduling for application i. Stage 1: posterior overlap elimination

$S \leftarrow 1$, $d_C \leftarrow 0$ // d_C: the cascaded delay
while $D_i > 0$ **do** // Delay limit not reached
  $d_{\max} \leftarrow 0$, $j_{\max} \leftarrow 0$
  for $j \in [S, n_i]$ **do** // Find $d_{\max}$
    $j' \leftarrow \texttt{NextTrans}(T_j)$
    $I_j \leftarrow \texttt{TranTime}(T_j) + \texttt{TranTime}(P_{j'}) + \texttt{Interval}(T_j, P_{j'})$
    if $I_j > d_{\max}$ **then**
      $j_{\max} \leftarrow j$, $d_{\max} \leftarrow I_j$
    end
  end
  if $d_{\max} \leq D_i$ **then**
    for $j \in [S, j_{\max}]$ **do** // Schedule T_j
      $d_j \leftarrow d_C$, $j' \leftarrow \texttt{NextTrans}(T_j)$
      if $d_j <$ $\texttt{Interval}(T_j, P_{j'})$ **then** // No overlap
        $d_j \leftarrow \texttt{Interval}(T_j, P_{j'})$
      end
      else // Overlap elimination
        $d_j \leftarrow \max\{d_C, \texttt{TranTime}(T_j) + \texttt{TranTime}(P_{j'}) + \texttt{Interval}(T_j, P_{j'})\}$
      $d_C \leftarrow d_j$
    end
    for $j \in (j_{\max}, n_i]$ **do**
      $d_j \leftarrow d_C$
    end
  end
  $D_i \leftarrow D_i - d_{\max}$, $S \leftarrow j_{\max} + 1$
end

C. Improvement of Computational Efficiency

The above scheduling algorithm, while ensuring the global scheduling optimality, incurs a large amount of computational overhead especially when D_i is large. To further improve the computational efficiency of transmission scheduling, we further develop efficient heuristic algorithms.

Our basic idea to address the possible overlap between transmissions is to divide the scheduling process into two stages. In the first stage, we iteratively look for the maximally allowed transmission delay within the application delay constraint, and then eliminate the transmission overlap due to such delay in a posterior manner. In the second stage when the delay constraint is being reached, we adopt a more conservative approach to ensure that all possible overlaps could be avoided prior to scheduling. Without loss of generality, we exclude the execution times $\{e_j^i\}$ of all the application methods from D_i, so that D_i being used in our algorithms only indicates the total amount of time allowed for data transmission.

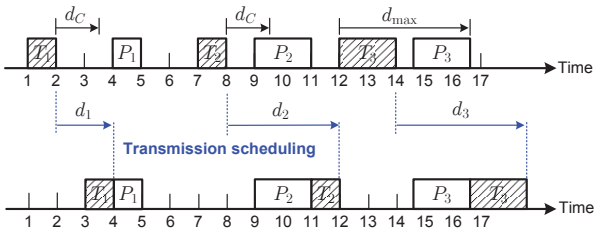
Algorithm 3: Greedy transmission scheduling for application i. Stage 2: prior overlap avoidance

$S \leftarrow j_{\max}$ // The value of $j_{\max}$ when Stage 1 completes
while $D_i > 0$ **do**
  $d_{\min} \leftarrow \infty$, $j_{\min} \leftarrow 0$
  for $j \in [S, n_i]$ **do** // Find $d_{\min}$
    $j' \leftarrow \texttt{NextTrans}(T_j)$
    $I_j \leftarrow \texttt{Interval}(T_j, P_{j'})$
    if $I_j < d_{\min}$ **then**
      $d_{\min} \leftarrow I_j$, $j_{\min} \leftarrow j$
    end
  end
  if $d_{\min} \leq D_i$ **then**
    for $j \in [S, n_i]$ **do**
      $d_j \leftarrow d_j + d_{\min}$ // Overlap avoidance
    end
  end
  $D_i \leftarrow D_i - d_{\min}$, $S \leftarrow j_{\min} + 1$
end

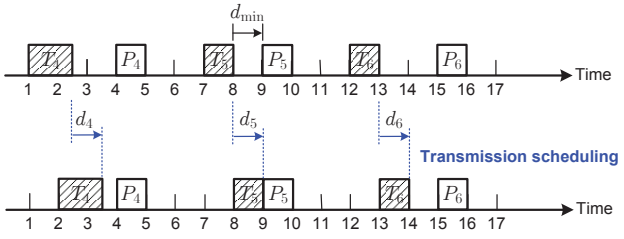
The first stage of transmission scheduling is described in Algorithm 2, where $\texttt{NextTrans}(T)$ returns the existing transmission next to T in the future, $\texttt{TranTime}(T)$ calculates the amount of time that T needs, and $\texttt{Interval}(T_1, T_2)$ calculates the elapsed time between T_1 and T_2. As shown in Figure 4, when T_j is delayed and overlaps with $P_{j'} = \texttt{NextTrans}(T_j)$, the amount of delay that T_j needs to eliminate such overlap will be $\texttt{TranTime}(T_j) + \texttt{TranTime}(P_{j'}) + \texttt{Interval}(T_j, P_{j'})$, so as to defer T_j to the completion of $P_{j'}$. In each round of scheduling, we look for the transmission $T_{j_{\max}}$ corresponding to the maximum amount of such delay ($d_{\max}$) over all the transmissions $\{T_j\}$. Then, any T_j ($j < j_{\max}$) could be effectively scheduled towards $\texttt{NextTrans}(T_j)$ without incurring any cascaded overlap to other transmissions after $T_{j_{\max}}$, and the overlap happened when scheduling such a T_j could hence be addressed by further deferring T_j to the end of $\texttt{NextTrans}(T_j)$.

Such scheduling process is illustrated in Figure 5(a) where $j_{\max} = 3$. $d_{\max}$ is directly applied as d_3, which reserves enough space for eliminating the overlap when scheduling T_1 and T_2. Since the existing cascaded delay d_C that is determined by scheduling other transmissions prior to T_1 is smaller than $\texttt{Interval}(T_1, P_1)$, d_1 could be simply assigned as $\texttt{Interval}(T_1, P_1)$ without incurring any overlap with P_1. T_2, on the other hand, will be scheduled to start after P_2, to eliminate the overlap between T_2 and P_2 caused by d_C.

When D_i is being reached and smaller than the $T_{j_{\max}}$ calculated in Stage 1, the scheduling algorithm switches to Stage 2, which is described in Algorithm 3. We look for the minimum value of $\texttt{Interval}(T_j, \texttt{NextTrans}(T_j))$ among all transmissions T_j in Stage 2, and then use this minimum interval ($d_{\min}$) to defer all transmissions prior to



(a) Stage 1: posterior overlap elimination



(b) Stage 2: prior overlap avoidance

Fig. 5. Illustrations of two-stage offline transmission scheduling algorithms

the corresponding $T_{j_{\min}}$. In this way, we avoid any overlap prior to $T_{j_{\min}}$, because only the interval between $T_{j_{\min}}$ and `NextTrans`$(T_{j_{\min}})$ is reduced to 0 after scheduling. As illustrated in Figure 5(b) where $j_{\min} = 5$ and $d_4 = d_5 = d_6 = d_{\min}$, no overlap is produced during transmission scheduling.

The computational complexity of transmission scheduling depends on the specific transmission times and intervals between transmissions. In particular, each round of transmission scheduling in either Stage 1 or Stage 2 has a complexity of $O(n)$ where n is the number of transmissions being scheduled. Both stages hence have an average-case time complexity of $O(n \operatorname{Log} n)$, while the worst-case complexity could be $O(n^2)$ if the scheduled delays d_j monotonically increase with j.

D. Flexible Energy-Delay Tradeoff Balancing

In practical MCC, users' preference between energy efficiency and application responsiveness may vary. For example, a user with a plugged-in device can spend more energy for timely application response, but may later want to preserve the battery power when the device is plugged off. In this section, we enable users to flexibly balance between these two aspects at run-time by adjusting the delay constraint of applications.

Algorithm 4: Transmission rescheduling with a new delay constraint D_i'

```
Δ_D ← D_i' − D_i          // Difference of delay constraint
if Δ_D > 0 then
    j_max = ScheduleStageOne(j_P + 1, n_i)
    ScheduleStageTwo(j_max + 1, n_i)
end
else
    for j ∈ [j_P + 1, n_i] do
        d_j = min{d_j − d_max, d_j + Δ_D}
    end
    for j ∈ [1, j_P] do
        if t_s^j − t_s^{j−1} > t̃_s^j − t̃_s^{j−1} then
            d_j = d_j − (t_s^j − t_s^{j−1}) + (t̃_s^j − t̃_s^{j−1})
        end
        else
            d_j = d_j − d_max
        end
    end
end
```

With a new delay constraint D_i' of application i, our rescheduling algorithm is described by Algorithm 4. For each transmission T_j ($j \in [1, n_i]$), we denote its originally scheduled start and end times as $\tilde{t}_s^j$ and $\tilde{t}_e^j$, and such times after rescheduling as t_s^j and t_e^j. We use j_P to indicate the separation of Stage 1 and Stage 2 in the original scheduling, such that transmissions $T_1, T_2, ..., T_{j_P}$ were scheduled in Stage 1, and others were scheduled as Stage 2. $j_{\max}$ and $d_{\max}$ indicate the same quantities as in Algorithm 2.

First, $\Delta_D = D_i' - D_i > 0$ indicates that the delay constraint of the application i is relaxed. Correspondingly, we further defer the data transmissions of i for more energy saving by involving more data transmissions into Stage 1 of scheduling. To do this, we simply re-execute the two-stage scheduling algorithms described in Section V-C over the set of data transmissions that were originally scheduled in Stage 2.

Second, $\Delta_D < 0$ indicates that the application should respond with shorter delay. For a transmission T_j that was originally scheduled in Stage 2, we could simply remove the cascaded delay $d_{\max}$ that was computed in Stage 1 from d_j. Since Stage 2 avoids transmission overlap prior to transmission scheduling, we are able to ensure that no additional transmission overlap could be incurred during such delay removal. Correspondingly, the same amount of cascaded delay has also to be removed from the transmissions in Stage 1. In particular, a transmission T_j could further reduce its scheduled delay d_j without incurring any overlap among other transmissions, if its interval between the previous transmission T_{j-1} after scheduling is larger than that before scheduling. In this case, such delay difference could be simply removed from d_j.

VI. Online Transmission Scheduling

In this section, we extend our proposed offline scheduling algorithms in Section V, by incorporating the run-time dynamics of application executions. More specifically, we predict the execution paths of applications in the future from the past history of their method invocations. Then, whenever a wireless transmission is requested, we probabilistically determine its transmission delay at run-time based on such prediction.

A. Prediction of Application Execution Path

According to existing studies [4], the transitions among program methods being executed by a mobile application at run-time could be precisely modeled by an order-k semi-Markov model, which captures and depicts the run-time dynamics of application executions. An order-k Markov renewal process representing the method transitions is a set of random variables $\{(s_m, t_m) : m \geq 0\}$, where state space $\mathbb{S} = \{M_1, M_2, ..., M_n\}$ is the set of methods of the application i. The random variable $t_{m+1} - t_m$ thus indicates the execution time of s_m. We consider that the invocation of a method s_m is determined by the previous k method invocations, which are indicated by the "composite state" $\tilde{s}_m$. We use a state transition probability matrix $\mathbf{T} \in \mathbb{R}^{n^k \times n}$. For any $p_{ij} \in \mathbf{T}$, $j \in \mathbb{S}$ represents a

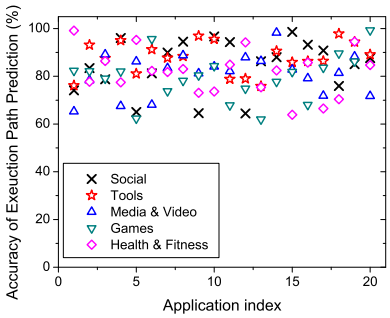


Fig. 6. Accuracy of predicting the application execution path

regular system state, and $i \in \mathbb{S}^k$ corresponds to a k-tuple of values in the original state space $\mathbb{S}$.

To further predict the application execution path, we consider that there are r absorbing states, i.e., the application will complete as soon as the method corresponding to an absorbing state is executed. Then, we consider an extended state transition probability matrix $\tilde{\mathbf{T}} \in \mathbb{R}^{n^k \times n^k}$ which depicts transitions among composite states, and denote a composite state $i = \{j_1, j_2, ..., j_k\}$ as absorbing if j_k is absorbing and all other states are non-absorbing. We then write $\tilde{\mathbf{T}}$ as

$$\tilde{\mathbf{T}} = \begin{bmatrix} \mathbf{Q} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix},$$

where the identity matrix $\mathbf{I}$ indicates transitions among absorbing composite states and $\mathbf{Q}$ indicates transitions among other states. Hence, the quantity $n_{ij} \in \mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}$ indicates the average number of transitions needed between two non-absorbing states i and j [7]. Furthermore, we could compute the average length of an execution path from a non-absorbing state i as $l_i \in \mathbf{L} = \mathbf{N} \cdot \mathbf{c}$ where $\mathbf{c} = [1, 1, ..., 1]'$.

To evaluate the accuracy of such prediction in practice, we crawled 100 real Android applications ranked as the most popular ones in five categories in the Google Play Store, and apply our approach to predict their execution paths over 50 runs with random user operations. Results in Figure 6 show that the prediction accuracy is above 75% for most of applications. It hence provides a solid foundation for further development of online transmission scheduling algorithms.

B. Probabilistic Transmission Scheduling

Based on the above formulation, we develop a probabilistic framework to adaptively schedule each transmission when it is requested by the application. The major challenge of such scheduling, however, is that it is difficult to estimate the impact of deferring the current transmission on the cumulative application responsiveness, without knowing the application execution pattern in the future. To overcome this challenge, our approach is to probabilistically estimate the cumulative transmission delay throughout the application execution, based on the above prediction of the length of application execution path in the future. More specifically, we consider a transmission T_j following the method execution M_j could be delayed for d_j, if d_j satisfies the following condition:

$$\mathbb{P}\left(l_j \cdot d_j + \sum_{u=j+1}^{j+l_j} t_e^u \leq D_i\right) \geq p, \tag{2}$$

where l_j is the predicted number of future method invocations, t_e^u is the execution time of method M_u to be invoked in the future, and p is the required probability for the application delay constraint to be met. Eq. (2) hence provides an upper bound of the transmission delay d_j, by considering that similar transmissions will happen in every method invocation in the future. The value of d_j is mainly determined by the execution times of application methods being invoked in the future. Let the random variable $X = \sum_{u=j+1}^{j+l_j} t_e^u$ indicate the cumulative execution time of these methods, we have

$$d_j = (D_i - F_X^{-1}(p))/l_j, \tag{3}$$

where $F_X(t)$ is the Cumulative Distribution Function (CDF) of X.

VII. Performance Evaluations

In this section, we evaluate the effectiveness of our proposed transmission scheduling approaches, by comparing with the following transmission scheduling schemes which ignore the run-time characteristics of mobile application executions.

- **Bundle transmission** [20]: the wireless transmissions are grouped together as bundles, without taking the performance requirements and delay constraints of mobile applications into account.
- **Fast dormancy** [11]: the mobile device quickly switches to the IDLE state after completion of data transmission, by sending a special message to the cellular network.
- **RSG** [17]: the wireless energy cost and application performance of workload offloading are considered in the same domain of transmission scheduling, with the assumption of independence among application method invocations. Run-time causality and heterogeneity of method invocations are ignored.

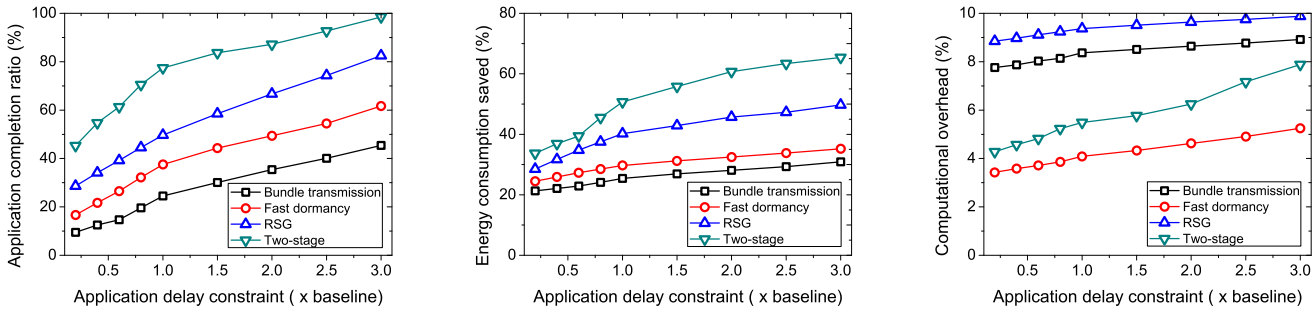
The following metrics are used in our evaluations. Each experiment is repeated multiple times with random input data or user operations for statistical convergence.

- **Application completion ratio**, the percentage of application executions that are completed within their specified delay constraints.
- **Amount of energy saved**, the percentage of local energy consumption saved by workload offloading. Energy consumed by both local computations and wireless transmissions is taken into account.
- **Computational overhead**, the mobile devices' local energy consumption for transmission scheduling. Particularly, such overhead is measured as the percentage of the energy consumption of application executions.

Note that in our evaluations, due to the heterogeneity of mobile application executions, it is difficult to directly measure and compare the response delay of multiple mobile applications with various delay constraints. Instead, as stated above, we use the completion ratio of application executions to indirectly evaluate the impact of traffic scheduling on the responsiveness of mobile applications. The longer a mobile application is deferred, the more likely that its execution is unable to complete within the given delay constraint.

A. Evaluation Setup

We evaluate the efficiency of our proposed approaches over the following multiple open-source smartphone applications, which exhibit significant heterogeneity in their computational complexity, input datasets, and responsiveness requirements. Hence, we believe that our results would be representative to the large population of mobile applications in practice.



(a) Application completion ratio (b) Amount of energy saved (c) Computational overhead

Fig. 7. Effectiveness of offline transmission scheduling

- **Firefox for Mobile** with v24.0 is one of the commonly used web browsers on mobile platforms.
- **Chess-Walk**[3] is a chess game with various modes.
- **Barcode Scanner**[4] is an application that scans products' barcodes and searches online for more information.

We adopt MAUI [2] for workload offloading decisions on application methods, and embed the implementation of workload offloading and transmission scheduling approaches into the source code of each application. Our operations of workload offloading then follow CloneCloud [1], such that a clone VM is maintained at our cloud server for each smartphone application being executed. Each time before an application method is invoked, our offloading engine is invoked first and determine whether this method will be offloaded for remote execution. The transmission of relevant program states required for such remote execution is then scheduled by our proposed algorithms.

Our evaluations use LG Nexus 4 smartphones running Android v4.4, and a Dell OptiPlex 9010 PC with an Intel i7-3770@3.4GHz CPU and 8GB RAM as the cloud server. The smartphones are connected to the server via UMTS cellular links. The local energy consumptions at smartphones are directly measured by a Monsoon power monitor, which provides DC power supply to the smartphones. We adopt the application profiling techniques in MAUI [2] to collect information about the method invocations and execution times.

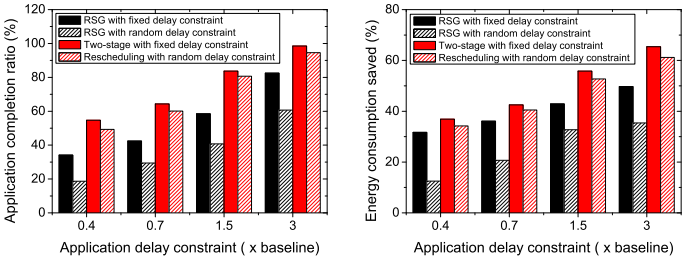
For each experiment, we preload input data to applications and execute them simultaneously. For Firefox, a randomly selected webpage with different interactive contents is loaded. For Chess-Walk, the steps of one pre-recorded game is replayed. For Barcode Scanner, the barcode of a randomly selected product is used. Each experiment is conducted 100 times with different input data for statistical convergence.

B. Effectiveness of Offline Transmission Scheduling

We first evaluate the effectiveness of our proposed offline transmission scheduling algorithms in Section V. In our experiments, each application is first analyzed using MAUI [2] to derive its global calling graph, based on which offloading decisions are made and the sequence of data transmissions for workload offloading is determined. For each experiment, we use the completion times of local application executions as the baseline, and vary the application delay constraint as different magnitudes of this baseline. Our experiments involve both the optimal scheduling algorithm proposed in Section V-B and the two-stage heuristic algorithms proposed in Section V-C. When the number of magnitudes over the baseline is smaller than 2.0, the optimal scheduling algorithm based on dynamic programming is used to ensure the scheduling optimality. Otherwise, two-stage heuristic algorithms are used to avoid incurring large amounts of computational overhead.

[3]https://gitorious.org/chesswalk

[4]http://code.google.com/p/zxing/



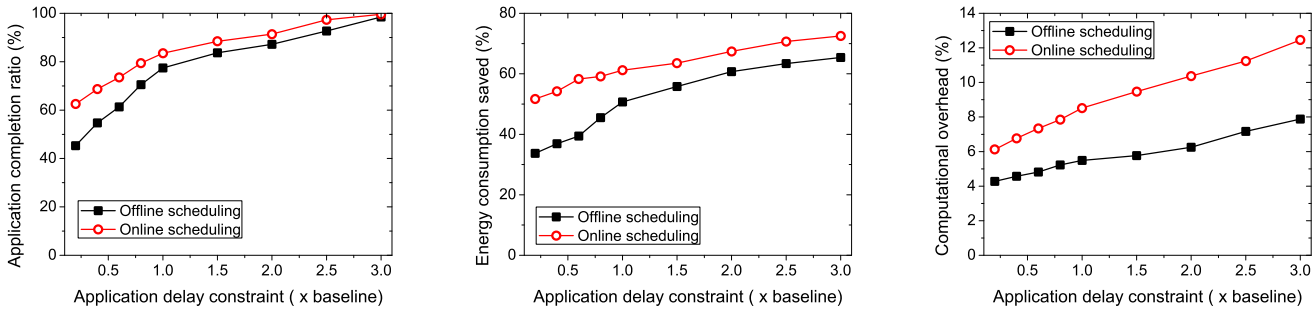
(a) Application completion ratio (b) Amount of energy saved

Fig. 8. Effectiveness of flexible energy-delay tradeoff balancing

The evaluation results are shown in Figure 7. By taking the causality among method invocations and subsequent wireless transmissions into account, our algorithms dramatically improve the application completion ratio. As shown in Figure 7(a), such ratio achieved by our algorithms is 25% higher than RSG which ignores the causality among method invocations, and more than 50% higher than other scheduling schemes ignoring the application delay constraints. Meanwhile, our scheduling algorithms significantly improve the energy efficiency of MCC operations. As shown in Figure 7(b), such amount of energy saving is related to the application delay constraint. When the constraint is tight (< 0.7x), the cascaded delay produced during scheduling incurs more overlap among transmissions, which degrades the network energy efficiency. The energy efficiency of our proposed algorithms in such cases is similar to that of RSG, but is still 40% better than those of Bundle transmission and Fast dormancy. Comparatively, when the constraint increases, our approaches could efficiently reduce the wireless energy cost by up to 60%.

We also evaluate the computational overhead of transmission scheduling. As shown in Figure 7(c), such overhead is less than 8% of the application execution overhead in all cases. Comparatively, the overhead of RSG could exceed 10% and increases the complexity of the joint optimization problem to be solved. Bundle transmission and Fast dormancy schemes have lower overhead due to their simpler scheduling process, but their ignorance of application execution characteristics also leads to lower efficiency of MCC operations.

We further evaluate the effectiveness of our proposed



(a) Application completion ratio (b) Amount of energy saved (c) Computational overhead

Fig. 9. Effectiveness of online probabilistic transmission scheduling

approach to flexible energy-delay tradeoff balancing, which is described in Section V-D. For each specified application delay constraint D, we randomly change it within the range $[0.5 \times D, 1.5 \times D]$ once every 20 method executions of an application run. The evaluation results in Figure 8 show that our proposed algorithm efficiently maintains a constant level of transmission scheduling efficiency when the delay constraint fluctuates, with a degradation less than 10%. Comparatively, such efficiency of RSG is seriously impaired by up to 50% due to its inflexibility of scheduling algorithm design.

C. Effectiveness of Online Transmission Scheduling

To evaluate the effectiveness of our proposed online transmission scheduling algorithms in Section VI, we use an order-3 semi-Markov model and compare the scheduling effectiveness of our proposed online and offline algorithms together. As shown in Figure 9, the difference of scheduling effectiveness between the online and offline algorithms is closely related to the application delay constraint. When the delay constraint is shorter (< 1.0x), the online algorithm improves the application completion ratio by up to 20% and energy efficiency by 25%, because of its incorporation of run-time dynamics of application executions and better prediction of application executions. Comparatively, the offline algorithm makes fixed scheduling decisions based on offline knowledge of application execution, which may largely deviate from the actual case at run-time. When the delay constraint increases, such difference is reduced to less than 10%, because a long delay constraint mitigates the impact of run-time execution dynamics.

Meanwhile, the evaluation results in Figure 9(c) show that the online scheduling algorithm incurs higher computational overhead. Especially when the delay constraint is larger than 1.5x, the online algorithm leads to 50% higher overhead but barely improves the scheduling effectiveness. These evaluation results, hence, suggest that online algorithms are more suitable for MCC scenarios with short application delay constraints, so as to minimize the impact of run-time heterogeneity of method executions on the scheduling decisions over short time periods.

VIII. Conclusions

In this paper, we propose novel schemes to adaptively balance the tradeoff between energy efficiency and delay performance of mobile applications in MCC. We take both causality and run-time dynamics of application executions into account, and develop offline and online algorithms to schedule the wireless transmissions incurred by workload offloading in MCC. The effectiveness of our proposed algorithms is evaluated by experiments over realistic smartphone applications.

References

[1] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the 6th Conference on Computer Systems*, pages 301–314, 2011.

[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 MobiSys*, pages 49–62, 2010.

[3] Y. Cui, S. Xiao, X. Wang, M. Li, H. Wang, and Z. Lai. Performance-aware energy optimization on mobile devices in cellular network. In *Proceedings of IEEE INFOCOM*, 2014.

[4] W. Gao, Y. Li, H. Lu, T. Wang, and C. Liu. On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *Proceedings of IEEE ICNP*, 2014.

[5] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen. Comet: Code offload by migrating execution transparently. In *Proceedings of USENIX OSDI*, pages 93–106, 2012.

[6] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards Wearable Cognitive Assistance. In *Proceedings of ACM MobiSys*, pages 68–81, 2014.

[7] J. G. Kemeny and J. L. Snell. *Finite markov chains*, volume 356. van Nostrand Princeton, NJ, 1960.

[8] 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 INFOCOM*, 2012.

[9] F. Liu, P. Shu, and J. Lui. AppATP: An energy conserving adaptive mobile-cloud transmission protocol. *IEEE Transactions on Computers*, 2015.

[10] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *Proceedings of Internet Measurement Conference (IMC)*, pages 137–150, 2010.

[11] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Top: Tail optimization protocol for cellular radio resource allocation. In *Proceedings of IEEE ICNP*, 2010.

[12] 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 MobiSys*, pages 43–56, 2011.

[13] J. S. Rellermeyer, O. Riva, and G. Alonso. Alfredo: An architecture for flexible interaction with electronic devices. In *Proceedings of the 10th International Middleware Conference*, pages 22–41, 2008.

[14] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[15] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proceedings of ACM MobiCom*, pages 85–96, 2010.

[16] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, and Y. Qu. eTime: energy-efficient transmission between cloud and mobile devices. In *IEEE INFOCOM*, pages 195–199. IEEE, 2013.

[17] 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 INFOCOM*, 2014.

[18] J.-H. Yeh, J.-C. Chen, and C.-C. Lee. Comparative analysis of energy-saving techniques in 3gpp and 3gpp2 systems. *IEEE Transactions on Vehicular Technology*, 58(1):432–448, 2009.

[19] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu. Energy-optimal mobile cloud computing under stochastic wireless channel. *IEEE Transactions on Wireless Communications*, 2013.

[20] B. Zhao, Q. Zheng, G. Cao, and S. Addepalli. Energy-aware web browsing in 3g based smartphones. In *Proceedings of ICDCS*, 2013.