Energy Delay Trade-off in Cloud Offloading for Mutli-core Mobile Devices

Zhefeng Jiang and Shiwen Mao
Department of Electrical and Computer Engineering
Auburn University, Auburn, AL 36849-5201 USA

***Abstract*—Cloud offloading is considered a promising approach to energy conservation and storage/computation enhancement for resource limited mobile devices. In this paper, we present a Lyapunov optimization based scheme for cloud offloading scheduling, as well as download scheduling for cloud execution output, for multiple applications running in a mobile device with a multi-core CPU. We derive an online algorithm and prove performance bounds for the proposed algorithm with respect to average power consumption and average queue length, which is indicative of delay, and reveal the fundamental trade-off between the two optimization goals.**

I. Introduction

There is a proliferation of mobile devices in recent years, such as smartphones and tablets, which are becoming more and more powerful with even multi-core CPUs. However, mobile devices still suffer from comparably limited resources. For example, the power of a smartphone comes at the cost of higher burden on the battery. As a result, although we are freed from a wireline data connection, we are still highly dependent on a power socket and charger. In addition, smartphones usually have relatively limited storage. With many apps, photos, and multimedia files recorded or cached, the internal storage space of our mobile devices can be easily depleted.

Cloud offloading has been recognized as an effective solution to the limited resource problem [1]. With offloading, we can store our photos and videos in the cloud and fetch it whenever it is needed. Furthermore, computation intensive tasks can also be offloaded to software clones in the cloud [2], so that most computation can be executed in the cloud to greatly reduce the burden on the mobile device. However, offloading data and computational tasks could involve considerable communications between mobile devices and cloud clones, which could consume a large amount of energy and incur extra delay. Hence, the decision between cloud offloading or local execution should be carefully made at each mobile device, taking into account the energy consumption and delay of various options, as well as the wireless network status.

In this paper, we study the problem of effective cloud offloading scheduling while considering downloading the output of cloud execution, for mobile devices with muti-core CPUs. We also consider task scheduling among cores of the CPU and frequency adaptation for the CPU considering both energy cost and user experience with respect to delay. Specifically, there are several trade-offs in making the optimal decisions. First, cloud offloading involves data transmissions from the mobile device to the cloud, as well as downloading the output of cloud execution, through a stochastic and unpredictable wireless channel. The energy efficiency of cloud offloading could be poor when the wireless coverage is weak. In such cases, energy may be conserved if we delay cloud offloading and downloading until the channel gets better, but at the cost of additional delays. Furthermore, cloud offloading may not be a good choice for applications with a large amount of offloading data or a large amount of output downloading, since transmitting the data over a wireless channel may consume considerable power and incur large delay, which offset the gains achieved by executing the task in the cloud. Similarly, energy can be conserved for local execution by reducing the CPU frequency, but at the cost of slower execution of the tasks. Motivated by these observations, we present a holistic formulation of the problem of optimal cloud offloading decision making for multiple applications running in a multi-core mobile device. The formulation takes into account the above trade-offs by incorporating the key control knobs, including CPU frequency and computation capability at the mobile device, offloading and downloading data volume of the applications, and the time-varying capacity and expected offloading power consumption of the wireless connection.

We then develop an effective solution algorithm to the formulated problem. The proposed scheduling algorithm is based on the Lyapunov optimizing framework [3], [4]. It dynamically schedules the tasks in the task queues for cloud offloading or local execution, downloads output from the cloud for offloaded tasks, and in the case of local execution, tunes the CPU frequency to balance energy consumption and delay, based on the current network condition and task queue backlogs. The proposed algorithm is inherently an *online algorithm*, meaning that it does not require information about the stationary distributions of the arrival and wireless channel processes, neither does any future application and network information. It makes decisions based on the current queue lengths and channel conditions. Such an online algorithm would be useful for real-time applications. We derive upper bounds on the average energy consumption and average queue length achieved by the proposed algorithm, which clearly reveal the trade-off between energy consumption and delay in optimal cloud offloading.

The rest of this paper is organized as follows. The system model and problem statement are presented in Section II. The proposed algorithm is developed in Section III. We review related work in Section IV. Section V concludes the paper.

978-1-4799-5952-5/15/$31.00 ©2015 IEEE

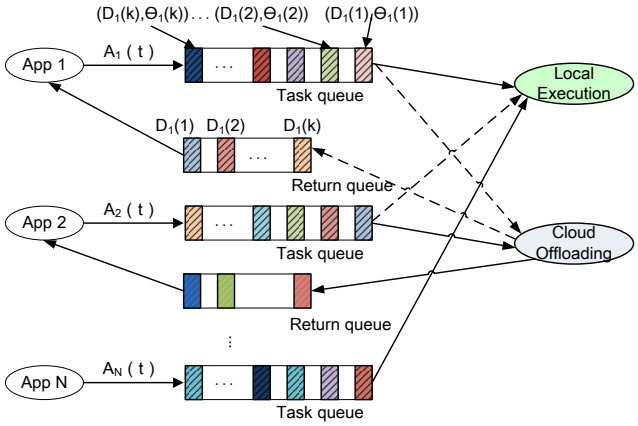

Fig. 1. The system model.

II. System Model and Problem Statement

A. System Model

The system model is illustrated in Fig. 1. We consider a mobile device having N applications running,[1] denoted as $\mathcal{N} = \{1, 2, \cdots, N\}$, among which $1 \leq N' \leq N$ applications, denoted as $\mathcal{N}'$, can be offloaded to the cloud. The tasks generated from each application are enqueued and processed in a First-In-First-Out (FIFO) manner. In addition, we assume that the arrival and execution of these tasks follow a discrete time-slotted system. In particular, the queue of tasks waiting to be processed for application i at the beginning of time slot t is denoted as $Q_i(t)$, and the overall queue lengths at the beginning of time slot t are denoted as $\mathcal{Q}(t) = \{Q_1(t), Q_2(t), \cdots, Q_N(t)\}$.

In time slot t, the tasks generated by applications are denoted as $\mathcal{A}(t) = \{A_1(t), A_2(t), \cdots, A_N(t)\}$, which can be regarded as new arrivals to $\mathcal{Q}(t)$. In this paper, we assume that each $A_i(t)$ is i.i.d. over time slots and the expectations of them, i.e., the average arrival rates, are denoted as $\vec{\lambda} \triangleq \mathbb{E}\{\mathcal{A}(t)\} = \{\lambda_1, \lambda_2, \cdots, \lambda_3\}$. The departing tasks from queue $\mathcal{Q}(t)$ is either for local execution or for offloading to the cloud, denoted as $\mathcal{B}(t) = \{B_1(t), B_2(t), \cdots, B_N(t)\}$. We also define the offloaded tasks in time slot t as $\mathcal{B}^O(t) = \{B_1^O(t), B_2^O(t), \cdots, B_N^O(t)\}$, where $B_i^O(t) = B_i(t)$ if the task of application i is offloaded, and $B_i^O(t) = 0$ otherwise.

In addition, we assume that for task k of application i, the computational complexity for local execution, $\theta_i(k,t)$ (i.e., the amount of computations required to accomplish the task), the data size for offloading, $D_i(k,t)$ (i.e., the amount of data transmitted for executing the task in the cloud), and the data size of the cloud execution output, $D_i^D(k,t)$ (i.e., the results to be returned to the mobile device), are all independent and identically distributed random variables (i.i.d). If the task cannot be offloaded to the cloud, then we have $D_i(k,t) = \infty$ and $D_i^D(k,t) = 0$. Alternatively, if the task can only be offloaded to the cloud, then we have $\theta_i(k,t) = \infty$.

[1]A multiple-thread application that enables parallel computing, can be treated as multiple applications.

When a task is offloaded, it is first processed by a server in the cloud and then the output for cloud execution is returned back to the mobile device. Hence, there is also a queue for the output data of cloud execution (e.g., at the access point or base station). Let $\mathcal{Q}^D(t)$ denote the returned output queue at the end of the time slot t, as shown in Fig. 1. We have

$$\mathcal{Q}^D(t) = \{Q_1^D(t), Q_2^D(t), \cdots, Q_N^D(t)\} \tag{1}$$

where $Q_i^D(t) = 0$ for $i \in \mathcal{N} \backslash \mathcal{N}'$, as there will be no output from cloud computing if the task cannot be offloaded. The arrival to the queue $\mathcal{Q}^D(t)$ can be denoted as

$$\mathcal{A}^D(t) = \{A_1^D(t), A_2^D(t), \cdots, A_N^D(t)\} \tag{2}$$

for an application i task that is to be offloaded, $|A_i^D(t)| = |B_i^O(t)|$. That is, if we ignore the time a cloud server takes to process the task, there is an increment of queue length in $Q_i^D(t)$ if a task in $Q_i(t)$ is offloaded to the cloud.

B. Local Execution Energy Consumption Model

For applications that are executed locally at the mobile device, most of the energy consumption comes from the CPU and the screen. As the screen energy consumption is largely dependent on the user habit, we do not take this part into account in this paper.[2] The energy consumption is thus mainly determined by the CPU operation.

In particular, the CPU energy consumption is proportional to v^2, where v is the CPU voltage [5]. Furthermore, the clock frequency of the CPU at time t, denoted as $f(t)$, is shown approximately linear to the CPU voltage v [5]. Therefore, the CPU power consumption in a CPU core occupied by application i in time slot t can be approximated as

$$\varepsilon_i(t) = \eta' \cdot f_i^2(t), \tag{3}$$

where η' is the energy coefficient determined by the CPU hardware architecture. As the energy consumption is linear with $f^2(t)$, energy can be saved by reducing the CPU frequency, which, however, will slow down the execution of the tasks.

A CPU schedule can be represented by $\{\alpha^L(t), \Theta(t)\}$, where $\alpha^L(t) \in \mathcal{N}$ is the set of applications being executed locally and $\Theta(t) = \{\Theta_1(t), \Theta_2(t), ..., \Theta_N(t)\}$ is the amount of computations a CPU core can offer to application i at time t, where $\Theta_i(t) = 0$, if $i \notin \alpha^L(t)$. Assuming that there are M cores in the CPU. We have $|\alpha^L(t)| \leq M$, i.e., the number of the parallel computing applications cannot exceed the number of cores in the CPU. For a given CPU architecture, the computational capability $\Theta_i(t)$ is usually linear with the CPU frequency. Hence, the CPU energy consumption at time t is also a quadratic function of $\Theta_i(t)$, i.e.,

$$\varepsilon_i(t) = \eta \cdot \Theta_i^2(t), \tag{4}$$

where η is the adjusted energy coefficient. The total energy consumption for local execution is $\varepsilon(t) = \sum_{i=1}^{N} \varepsilon_i(t)$.

[2]It may be annoying to dynamically adjust the display size, resolution, or brightness during the execution of an application. We simply assume some constant amount of energy consumption associated with this part.

C. Offloading Energy Consumption Model

For applications that can be offloaded to the cloud, we make the following assumptions. First, we assume that a software clone has already been associated with each application in the cloud to support cloud computing [6], such that only the latest use generated data, application status updates, and cloud execution output, refereed to as *offloading data*, need to be transmitted between the mobile device and the cloud.

Second, we focus on the channel models associated with the wireless interfaces and ignore the delay and energy consumption in the cloud, which are justifiably minor issues comparing to that on the mobile device side. It is typical for a smartphone to choose one of the mobile networks (e.g., 2G, 3G, LTE, and WiFi) and the corresponding data rate is determined by the operator and the baseband chip configuration. We adopt the network selection algorithm proposed in [7] to choose between a cellular network and WiFi, and focus on the task scheduling problem in this paper.

Let $\omega_O(t)$ be the wireless link data rate from the mobile device to the cloud, and $\omega_D(t)$ the data rate from the cloud to the mobile device. An offloading decision is denoted as $\alpha^O(t) \in \{\mathcal{N}', '\text{idle}'\}$. That is, the device can choose to offload a task from one of the eligible queues or remain idle (i.e., to choose local execution). Then, the expected energy consumption is denoted as $p_O(t)$. Similarly, the decision for downloading the cloud execution output can be denoted as $\alpha^D(t) \in \{\mathcal{N}', '\text{idle}'\}$, and the expected energy consumption is denoted as $p_D(t)$.

D. Queuing and the Overall Energy Consumption Model

As discussed, energy can be conserved by optimizing the execution decision for the application tasks, i.e., local execution or offloading to the cloud. For local execution, energy can be saved by reducing the CPU frequency (i.e., running the application at a lower speed, which leads to a smaller $\Theta(t)$). For offloading, energy can be saved by only using good channels for transmission of offloading data and receiving the cloud output. There maybe an additional delay to wait for the channel to get better. If we aggressively save power by these means, the applications will suffer from large delays; the lengths of the task queues may increase to very high levels and the system may become unstable. We need to balance energy saving and delay, which is indicated by the task queue length.

Define $P(t) = \varepsilon(t) + p_O(t) + p_D(t)$. Based on the local execution and offloading energy consumption models, the overall energy consumption of the mobile device can be derived as follows.

$$\bar{P} \triangleq \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\varepsilon(t) + p_O(t) + p_D(t)\}. \tag{5}$$

We define the average task and output queue length, denoted as $\bar{Q}$, for evaluation of the energy-queue trade-off as

$$\bar{Q} \triangleq \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i=1}^{N} \mathbb{E}\{Q_i(t) + Q_i^D(t)\}, \tag{6}$$

where $Q_i(t)$ is the task queue length for application i at time t, and $Q_i^D(t)$ is the cloud output queue length for application i at time t. We consider the system to be stable if the average queue length is bounded, i.e., the limit in (6) exists.

The dynamics of the task queue backlog $Q_i(t)$ can be written as

$$Q_i(t+1) = \max\{Q_i(t) + A_i(t) - B_i(t), 0\}, \forall\, i, \tag{7}$$

where $B_i(t)$ is the service rate at time t defined as follows.[3]

$$B_i(t) = \begin{cases} \arg\max_{\{b\}} \left\{\sum_{k=1}^{b} \theta_i(k) \le \Theta_i(t)\right\}, \\ \qquad\qquad \text{if } i \in \alpha^L(t) \\ \arg\max_{\{b\}} \left\{\sum_{k=1}^{b} D_i(k) \le \omega_O(t)\right\}, \\ \qquad\qquad \text{if } i \in \alpha^O(t) \\ 0, \qquad\qquad \text{otherwise.} \end{cases} \tag{8}$$

Note that $\alpha^L(t)$ and $\alpha^O(t)$ should not point to the same application i, as it is inefficient to both offload and locally execute the same application task at the same time. If $i \in \alpha^L(t)$, the task queue of application i is executed locally and $B_i(t)$ is the maximum number of tasks can be executed locally at time slot t. If $i \in \alpha^O(t)$, the tasks of application i are offloaded to the cloud and $B_i(t)$ is the maximum number of tasks can be offloaded at this time slot.

Similarly, the dynamics of the cloud output queue backlog $Q_i^D(t)$ can be written as

$$Q_i^D(t+1) = \max\{Q_i^D(t) + A_i^D(t) - B_i^D(t), 0\}, \forall\, i \in \mathcal{N}', \tag{9}$$

where $B_i^D(t)$ is the service rate at time i for the cloud output queue defined as

$$B_i^D(t) = \begin{cases} \arg\max_{\{b\}} \left\{\sum_{k=1}^{b} D_i^D(k) \le \omega_D(t)\right\}, \\ \qquad\qquad \text{if } i \in \alpha^D(t) \\ 0, \qquad\qquad \text{otherwise.} \end{cases} \tag{10}$$

If $i \in \alpha^D(t)$, the cloud output queue i is downloaded and $B^D(t)$ is the maximum number of tasks that can download their cloud output at this time slot.

E. Problem Statement

For a mobile device, it makes task scheduling decisions about offloading and local execution at the beginning of each slot. It then makes decisions for downloading the return data of cloud execution for the next slot at the end of current time slot. The objective of mobile devices is to keep all the queues stable and to minimize the overall energy consumption. The scheduling problem can be formulated as

$$\min: \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\varepsilon(t) + p_O(t) + p_D(t)\} \tag{11}$$

$$\text{s.t. } \alpha^L(t) \cap \alpha^O(t) = \emptyset, \text{ for all } t \tag{12}$$

$$\left|\alpha^L(t)\right| \le M, \text{ for all } t \tag{13}$$

$$\bar{Q} < \infty, \tag{14}$$

[3]We assume that the duration of a time slot is large enough such that any task can be executed locally, offloaded to the cloud, or with output downloaded from the cloud in less than one time slot. This can be achieved by choosing a suitable time slot duration or by partitioning big tasks into smaller ones.

where Constraint (12) forbids a task to be both executed locally and offloaded to the cloud in the same time slot, Constraint (13) is the limitation forced by the number of cores in the CPU, and Constraint (14) ensures stability of the task and output queues. The optimal solution to the problem consists of cloud offloading or local execution decisions for each time slot t (i.e., $\alpha^L(t)$ and $\alpha^O(t)$) and the optimized CPU computation capability $\Theta(t)$ for each time slot t, which translates to the optimal CPU clock frequency f as discussed in Section II-B (configured as in (25)).

III. Task Scheduling Algorithm for Mobile Users

In this section, we present a task scheduling algorithm based on the Lyapunov optimization framework [3]. This algorithm requires no information about the stationary distributions of the arrival and wireless channel processes; it only requires information on the current queue lengths and the current channel conditions. Such an *online algorithm* property is useful for real-time applications [4], [8], [9].

A. Lyapunov Optimization based Solution Algorithm

To present the proposed algorithm, we first define a Lyapunov function $L(Q(t))$ as in [3].

$$L(Q(t)) \triangleq \frac{1}{2}\sum_{i=1}^{N} Q_i^2(t) + \frac{1}{2}\sum_{i=1}^{N}\{Q_i^D(t)\}^2, \tag{15}$$

where $L(Q(0)) = 0$. If all the queue lengths are small, then $L(Q(t))$ will be small; if at least one queue is congested, then $L(Q(t))$ will become large. Since there is a finite number of applications running on the mobile device, $L(Q(t))$ being bounded is equivalent to the notion that the system is stable.

Since $L(Q(0)) = 0$, for $L(Q(t+1))$, we have

$$\mathbb{E}\{L(Q(t+1))\} = \mathbb{E}\left\{\sum_{k=0}^{t}[L(Q(k+1)) - L(Q(k))]\right\}$$

$$= \sum_{k=0}^{t}\mathbb{E}\{L(Q(k+1)) - L(Q(k))|Q(k)\} = \sum_{k=0}^{t}\Delta(L(k)),$$

where $\Delta(L(t))$ is the *drift* defined as [10]

$$\Delta(L(t)) \triangleq \mathbb{E}\{L(Q(t+1)) - L(Q(t))\,|Q(t)\}. \tag{16}$$

We can minimize $\Delta(L(t))$ to maintain a low expectation for $L(Q(t))$. It follows (7) that

$$Q_i^2(t+1) \le \{Q_i(t) + A_i(t) - B_i(t)\}^2. \tag{17}$$

For $i \in \alpha^O(t)$, we have

$$\{Q_i^D(t+1)\}^2 \le \{Q_i^D(t) + B_i(t) - B_i^D(t)\}^2. \tag{18}$$

For $i \notin \alpha^O(t)$, we have

$$\{Q_i^D(t+1)\}^2 \le \{Q_i^D(t) - B_i^D(t)\}^2. \tag{19}$$

Substituting (17), (18), and (19) into (16), we derive the drift (20) as given on top of the next page. In (20), φ denotes the term in the expectation operator and $\Phi = \frac{1}{2}\sum_{i=1}^{N}\mathbb{E}\{\{A_i(t) - B_i(t)\}^2 + \{B_i^O(t) - B_i^D(t)\}^2\}$. Note that $B_i^O(t) = 0$ for $i \notin \alpha^O(t)$ and $B_i^O(t) = B_i(t)$ for $i \in \alpha^O(t)$. If the arrival rate and service rate of each queue is bounded, which is true for stable systems, then Φ is bounded.

As in [3], we obtain the *drift-plus-penalty*, defined as $\Delta(L(t)) + V_p \cdot \mathbb{E}\{P(t)\}\}$, by scaling the energy consumption with a positive coefficient V_p. The parameter V_p indicates the user's emphasis on energy consumption. Following (20), the upper bound of the *drift-plus-penalty* can be derived as

$$\Delta(L(t)) + V_p \cdot \mathbb{E}\{P(t)\} \le \Phi + \mathbb{E}\left\{\varphi + V_p \cdot P(t)\right\}. \tag{21}$$

To minimize the drift-plus-penalty, we can instead minimize $\{\varphi + V_p \cdot P(t)\}$ at every time slot, which only requires the current information on queue lengths, channel conditions, and the price for offloading.

Since there are M cores in the CPU of the mobile device, only M application can be executed by the CPU in each time slot. We assume that only one application can be offloaded at each time slot (through the single active wireless connection). We can derive the minimization expression as given in (22) on top of the next page. The first term in (22), $\sum_{i=1}^{N} Q_i(t)A_i(t)$, only depends on the current queue lengths and arrival rates. It does not affect the offloading downloading decision for this time slot. We need to minimize the second term $H_1 = V_p p_D(t) - Q_i^D(t)B_i^D(t)|_{i\in\alpha^D(t)}$, as a function of $\alpha^D(t)$, and the third term $H_2 = V_p\varepsilon(t) - \sum_{i\in\alpha^L(t)} Q_i(t)B_i(t) + \{V_p p_O(t) - (Q_i(t) - Q_i^D(t))B_i^O(t)|_{i\in\alpha^O(t)}\}$, as a function of $\alpha^L(t)$, $\alpha^O(t)$, and $\Theta(t)$.

Notice that in $\min\{H_1\}$, with the expectation of power consumption and offloading data, we need to find a proper $\alpha^D(t)$ that minimizes $-Q_i^D(t)B_i^D(t)$ in order to minimize the following function.

$$\xi_i^D = V_p p_D(t) - Q_i^D(t)B_i^D(t). \tag{23}$$

This can be done by evaluating (23) for every application in $\mathcal{N}'$ to find the application i having the smallest ξ_i^D. Recall that $B_i^D(t)$ is defined in (10). For a given downlink capacity $\omega_D(t)$, tasks with smaller data size and longer queue length tend to have a smaller $-Q_i^D(t)B_i^D(t)$. Note that $V_p p_D(t) - Q_i^D(t)B_i^D(t)|_{i\in\alpha^D(t)} = 0$ when $\alpha^D(t) =$ 'idle'. Thus a task will be offloaded in time slot t only when $\min\{V_p p_D(t) - Q_i^D(t)B_i^D(t)\} < 0$, meaning the channel condition is good or at least one of the task queues is long.

For the other term H_2, we need to minimize it by tuning $\alpha^L(t)$, $\alpha^O(t)$, and$\Theta(t)$. The term $V_p\varepsilon_i(t) - Q_i(t)B_i(t)$ can be rewritten as

$$\begin{aligned} &V_p\varepsilon_i(t) - Q_i(t)B_i(t) \\ &= V_p\eta\Theta_i^2(t) - Q_i(t)\cdot\arg\max_{\{b\}}\left\{\sum_{k=1}^{b}\theta_i(k,t) \le \Theta_i(t)\right\} \\ &\cong V_p\eta\Theta_i^2(t) - Q_i(t)\frac{\Theta_i(t)}{\bar{\theta}_i(t)}, \end{aligned} \tag{24}$$

where $\bar{\theta}_i(t) = \frac{1}{Q_i(t)}\sum_{k=1}^{Q_i(t)}\theta_i(k,t)$. We can derive the approximate minimum value $V_p\varepsilon(t) - Q_i(t)B_i(t)$ subject to the

$$\Delta(L(t)) \le \Phi + \mathbb{E}\left\{\sum_{i\notin\alpha^O(t)} Q_i(t)(A_i(t)-B_i(t)) - Q_i^D(t)B_i^D(t)\right\} + \mathbb{E}\left\{\{Q_i(t)A_i(t) - (Q_i(t)-Q_i^D(t))B_i(t) - Q_i^D(t)B_i^D(t)\}|_{\{i\in\alpha^O(t)\}}\right\}$$
$$= \Phi + \mathbb{E}\{\varphi\}. \tag{20}$$

$$\min\{\varphi + V_p P(t)\} = \min\left\{\sum_{i=1}^N Q_i(t)A_i(t) - \sum_{i=1}^N Q_i^D(t)B_i^D(t) - \sum_{i\notin\alpha^O(t)} Q_i(t)B_i(t) - (Q_i(t)-Q_i^D(t))B_i^O(t)|_{i\in\alpha^O(t)} + V_p P(t)\right\}$$
$$= \sum_{i=1}^N Q_i(t)A_i(t) + \min\left\{V_p p_D(t) - \sum_{i=1}^N Q_i^D(t)B_i^D(t) + V_p\varepsilon(t) - \sum_{i\notin\alpha^O(t)} Q_i(t)B_i(t) + V_p p_O(t) - (Q_i(t)-Q_i^D(t))B_i^O(t)|_{i\in\alpha^O(t)}\right\}$$
$$= \sum_{i=1}^N Q_i(t)A_i(t) + \min\left\{V_p p_D(t) - Q_i^D(t)B_i^D(t)|_{i\in\alpha^D(t)}\right\} + \min\left\{V_p\varepsilon(t) - \sum_{i\in\alpha^L(t)} Q_i(t)B_i(t) + \{V_p p_O(t) - (Q_i(t)-Q_i^D(t))B_i^O(t)|_{i\in\alpha^O(t)}\}\right\}$$
$$= \sum_{i=1}^N Q_i(t)A_i(t) + \min\{H_1\} + \min\{H_2\}. \tag{22}$$

CPU computation capability $\Theta_i(t)$ as

$$\xi_i^L(t) = -\frac{Q_i^2(t)}{4V_p\eta\bar{\theta}_i^2(t)}, \text{ when } \Theta_i(t) = \frac{Q_i(t)}{2V_p\eta\bar{\theta}_i(t)}. \tag{25}$$

Similarly, we can evaluate (25) for all the applications in $\mathcal{N}$ and find the minimizer. Since the computational capability of the CPU cannot be increased indefinitely, we set an upper bound for the CPU power, e.g., 10 W in this paper.

For the term $\{V_p p_O(t) - (Q_i(t) - Q_i^D(t))B_i^O(t)|_{i\in\alpha^O(t)}\}$, we can minimize it by tuning $\alpha^O(t)$. Denoting

$$\xi_i^O = V_p p_O(t) - (Q_i(t) - Q_i^D(t))B_i^O(t) < 0, \tag{26}$$

an application $i \in \mathcal{N}'$ with smaller offloading data size and greater $Q_i(t) - Q_i^D(t)$ will achieve a smaller ξ_i^O. Also note that $\{V_p p_O(t) - (Q_i(t) - Q_i^D(t))B_i^O(t)|_{i\in\alpha^O(t)}\} = 0$ when $\alpha_i^O =' idle'$. Thus a task can be offloaded only when $\xi_i^O < 0$.

Then the $\min\{H_2\}$ term can be rewrite as

$$\min\{H_2\} = \min\left\{\sum_{i\in\alpha^L(t)} \xi_i^L + \xi_j^O|_{j\in\alpha^O(t),\alpha^O(t)\cap\alpha^L(t)=\emptyset}\right\}.$$

According to the above evaluation, the problem becomes

$$\sum_{i=1}^N Q_i(t)A_i(t) + \min\{H_1\} + \min\{H_2\}$$
$$= \sum_{i=1}^N Q_i(t)A_i(t) + \min\left\{\xi_i^D\right\} + \min\left\{\sum_{i\in\alpha^L(t)} \xi_i^L + \xi_j^O|_{j\in\alpha^O(t),\alpha^O(t)\cap\alpha^L(t)=\emptyset}\right\}, \tag{27}$$

where ξ_i^D, ξ_i^L, and ξ_j^O are defined in (23), (25) and (26), respectively. We also have $\alpha^O(t)\cap\alpha^L(t) = \emptyset$, since the same application cannot be executed locally and offloaded to the cloud in the same time slot. The proposed task scheduling algorithm is presented in Algorithm 1. In Algorithm 1, all computations except step 2 are simple operations. For step 2, it can be relaxed to be solved with the Hungarian algorithm [11] with complexity $O(N*(M+1)^2)$ if $(M > N)$, or $O(M*(N+1)^2)$ otherwise.

Algorithm 1: Task Scheduling Algorithm

1 Update all the task queues and estimate wireless link capacities at the beginning of time slot t ;
2 Find the minimum combination of $\sum_{i\in\alpha^L(t)} \xi_i^L + \xi_j^O$, where $\alpha^O(t)\cap\alpha^L(t) = \emptyset$ and $j \in \mathcal{N}'$;
3 **if** $\xi_j^O < 0$ **then**
4 Offload application j to the cloud ;
5 **end**
6 **for** $i \in \alpha^L(t)$ **do**
7 **if** $\xi_i^L < 0$ **then**
8 Execute application i locally, with CPU capacity $\Theta_i(t) = \frac{Q_i(t)}{2V_p\eta\bar{\theta}_i(t)}$;
9 **end**
10 **end**
11 Find the minimum ξ_i^D;
12 **if** $\xi_i^D < 0$ **then**
13 Fetch the output data for application i from the cloud ;
14 **end**

B. Performance Analysis

Following the framework of Lyapunov optimization [3], we derive the upper bounds for the expected average power consumption and the expected average queue length achieved by the proposed algorithm, which are summarized in the following theorem. The proof is omitted for lack of space.

Theorem 1. *Assume that the arrival rate of tasks $\vec{\lambda}$ is strictly within the system capacity region. That is, the system can maintain stability under certain $\{\alpha^L(t), \alpha^O(t), \alpha^D(t), \Theta(t)\}$. Then the bounds on average energy consumption and queue length under Algorithm 1 can be written as*

$$\limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T-1} E\{P(t)\} < P^* + \frac{\Phi}{V_p} \tag{28}$$

$$\limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T-1}\sum_{i=1}^{N} E\{Q_i(t) + Q_i^D(t)\} < \frac{1}{\epsilon}(\Phi + V_p P), \tag{29}$$

where P^* is the minimum energy consumption a stable system

can achieve, P is the energy consumption under the proposed algorithm, and $\epsilon > 0$ is the distance between the data arrival rate vector $\vec{\lambda}$ and the system capacity region under the proposed algorithm.

Theorem 1 demonstrates the trade-off between energy consumption and queue length (or, delay). The upper bound of the average energy consumption is $O(1/V_p)$ and the upper bound of the average queue length is $O(V_p)$. Therefore these are conflicting objectives. We can tune V_p to flexibly trade off between energy consumption and queue length. When the power supply is not so limited (e.g., a charger is available), the user can increase V_p to reduce the queue length (and thus delay) and enjoy better quality of experience (QoE). On the other hand, if the power constraint is stringent (e.g., the mobile device is running out of battery and no charger is available), the user can decrease V_p to save energy at the expense of longer average queue length and larger delay.

IV. Related Work

Cloud offloading is regarded as an effective solution to save energy, extend storage spaces, and enable computation intensive applications at mobile devices [1]. There have been many prior work addressing the various design issues of cloud computing to fully harvest its potential [13]–[15]. In particular, considerable recent works have focused on building the framework of enable mobile computation offloading [2], [16], while many other works have focused on backing up data and applications to extend the storage space of mobile devices. Both computation offloading and data/application back up involve considerable energy consumption for data transmission between mobile devices and the cloud.

Researchers have started to investigate the energy cost of offloading. For example, the bandwidth and energy costs of cloud computing were investigated in [6]. In [17], the authors proposed a dynamic offloading algorithm to save energy by offloading some components of an application to the cloud. However, the stochastic characteristics of applications and network dynamics have not been taken into account in this work. The authors of [14], [18] proposed an energy-optimal mobile computing framework under stochastic wireless channels, while considering the single application scenario. In [10], an energy-efficient transmission algorithm between the cloud and mobile devices was presented based on the Lyapunov optimizing framework [3]. However, the local computation resources in the mobile devices has not been fully utilized, and it doesn't consider downloading the cloud computing output.

This work was motivated by the above interesting works to investigate the energy-delay trade-off in cloud offloading with a Lyapunov optimization approach. We explicitly considered the stochastic user and application behaviors and addressed the more challenging case of multiple applications, thus extending the work in [14], [18]. This work also extended [10] by considering multi-core CPUs and fully utilizing the local computing capability, making offloading decisions based on both task queues and queues for downloading the output of cloud computing. As in [10], the online operation of the proposed scheme makes it highly suitable for real-time applications.

V. Conclusions

In this paper, we proposed a scheduling scheme for energy-efficient cloud offloading for muti-core mobile devices. Based on Lyapunov optimization, we developed an online algorithm that does not require information about stationary distribution of applications and the network condition, making it amenable to real-time implementation for practical scenarios. We proved theoretical bounds for the proposed algorithm.

Acknowledgment

This work is supported in part by the NSF under Grant CNS-1247955, and the Wireless Engineering Research and Education Center (WEREC) at Auburn University.

References

[1] Y. Xu and S. Mao, "A survey of mobile cloud computing for rich media applications," *IEEE Wireless Commun.*, vol. 20, no. 3, pp. 46–53, June 2013.

[2] S. Kosta and et al., "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM 2012*, Orlando, FL, Mar. 2012, pp. 945–953.

[3] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool Publishers, 2010.

[4] Y. Huang, S. Mao, and R. M. Nelms, "Adaptive electricity scheduling in microgrids," *IEEE Trans. Smart Grid*, vol. 5, no. 1, pp. 270–281, Jan. 2014.

[5] T. D. Burd and R. W. Brodersen, "Processor design for portable systems," *Kluwer J. VLSI Signal Process. Syst.*, vol. 13, no. 2/3, pp. 203–221, Aug. 1996.

[6] M. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? The bandwidth and energy costs of mobile cloud computing," in *Proc. IEEE INFOCOM 2013*, Turin, Italy, Apr. 2013, pp. 1285–1293.

[7] A. J. Nicholson and et al., "Improved access point selection," in *Proc. ACM MobiSys'06*, Uppsala, Sweden, June 2006, pp. 233–245.

[8] Y. Wang, S. Mao, and R. M. Nelms, "An online algorithm for optimal real-time energy distribution in smart grid," *IEEE Trans. Emerg. Topics Comput.*, vol. 1, no. 1, pp. 10–21, July 2013.

[9] Z. Jiang and S. Mao, "Online channel assignment, transmission scheduling, and transmission mode selection in multi-channel full-duplex wireless LANs," in *Proc. WASA 2015*, Qufu, China, Aug. 2015, pp. 1–10.

[10] P. Shu and et al., "eTime: Energy-efficient transmission between cloud and mobile devices," in *Proc. IEEE INFOCOM 2013*, Turin, Italy, Apr. 2013, pp. 195–199.

[11] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. [Online]. Available: http://dx.doi.org/10.1002/nav.3800020109

[12] J. Huang and et al., "A close examination of performance and power characteristics of 4G LTE networks," in *Proc. ACM MobiSys'12*, Low Wood Bay, UK, June 2012, pp. 225–238.

[13] W. Zhang, Y. Wen, Z. Chen, and A. Khisti, "QoE-driven cache management for HTTP adaptive bit rate streaming over wireless networks," *IEEE Trans. Multimedia*, vol. 15, no. 6, pp. 1431–1445, Oct. 2013.

[14] W. Zhang and et al., "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sept. 2013.

[15] Y. Jin, Y. Wen, H. Hu, and M.-J. Montpetit, "Reducing operational costs in cloud social TV: An opportunity for cloud cloning," *IEEE Trans. Multimedia*, vol. 16, no. 6, pp. 1739–1751, Oct. 2014.

[16] B.-G. Chun and et al., "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. EuroSys'11*, Salzburg, Austria, Apr. 2011, pp. 301–314.

[17] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Trans. Wireless Commun.*, vol. 11, no. 6, pp. 1991–1995, June 2012.

[18] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE INFOCOM 2012*, Orlando, FL, Mar. 2012, pp. 2716–2720.