Energy Aware Offloading for Competing Users on a Shared Communication Channel

Erfan Meskar, Terence D. Todd and Dongmei Zhao
Department of Electrical and Computer Engineering
McMaster University
Hamilton, Ontario, CANADA
Email: {meskare,todd,dzhao}@mcmaster.ca

George Karakostas
Department of Computing and Software
McMaster University
Hamilton, Ontario, CANADA
Email: karakos@mcmaster.ca

***Abstract*—This paper considers a set of mobile users that employ cloud-based *computation offloading*. In order to execute jobs in the cloud however, the user uploads must occur over a base station channel that is shared by all of the uploading users. Since the job completion times are subject to hard deadline constraints, this restricts the feasible set of jobs that can be processed. The system is modelled as a competitive game in which each user is interested in minimizing its own energy consumption. The game is subject to the real-time constraints imposed by the job execution deadlines, user specific channel bit rates, and the competition over the shared communication channel. The paper shows that for a wide range of parameters, a game where each user independently sets its offloading decisions always has a pure Nash equilibrium, and a Gauss-Seidel-like method for determining this equilibrium is introduced. Results are presented that illustrate that the system always converges to a Nash equilibrium using the Gauss-Seidel method. Data is also presented that show the number of iterations required, and the quality of the solutions. We find that the solutions perform well compared to a lower bound on total energy performance.**

***Index Terms*—Cloud computing, shared communications, computation offloading, game theory**

I. Introduction

Mobile cloud computing (MCC) is already starting to revolutionize mobile device operation. In addition to its other benefits, MCC can reduce mobile user energy requirements by moving computational tasks and data storage functions away from the user, and onto infrastructure-based cloud servers. This enables the users to benefit from applications that would otherwise tax the resources of the user if they were to be run locally [1]. This functionality is being enabled, in part, by virtualization methods that permit cloud-based servers to run applications on behalf of their mobile clients [2]. According to a recent study by Cisco Inc., it is expected that mobile cloud traffic will increase by a factor of twelve over the next five years, with a compound yearly growth rate of over 60 percent. Cloud based application support is projected to account for 50 percent of total mobile data traffic by 2018 [3].

The commercial success of mobile cloud computing has motivated a wide variety of recent research. In reference [4], three MCC architectures are discussed: centralized clouds, cloudlets and ad-hoc cloud configurations. The work in [5] investigated issues involving the support of MCC in heterogeneous networks. These, and many other studies have illustrated the value of mobile cloud computing. This includes work that leverages the increased cloud-based storage capacity and remote processing, while improving mobile user energy efficiency, resulting in improved battery lifetime [6], [7], [8].

In this paper we consider the use of computation offloading, where mobile user energy consumption is improved by offloading job execution to remote cloud servers, rather than performing the computations locally. It has been shown that remote application execution can significantly improve mobile battery lifetime in these types of situations [9]. Computation offloading exploits the use of cloud-based servers that have significantly more resources than that of a typical mobile user. There are also studies that have considered computation offloading from an application execution viewpoint, where jobs are partitioned into multiple local and remote execution components [10]. In this case, the appropriate job partitions must be selected for local and remote execution.

Reference [11] proposes an architecture known as MAUI, which controls computation offloading for runtime .NET applications. Using .NET features, MAUI profiles the application and formulates computation offloading as a linear program (LP). Reference [12] proposes a similar architecture for Android applications. Recent work has also proposed a variety of application offloading mechanisms [12][13][14][15]. Various cloud-enabled platforms have also been proposed, such as cloudlet servers [13] and cloud clones [12]. In the latter case, a mobile user is associated with a system-level cloud-hosted clone that runs in a virtual machine, and executes jobs on behalf of the mobile user.

Game theory has been used to model mobile cloud computing. In reference [16], multiple service providers cooperatively offer mobile services, and a competitive game was used to share revenue. Reference [1] reduces energy consumption at both the server and users, so that sustainability is achieved.

A congestion-based game and optimization framework was used, where the mobile users are players and the strategy is to select servers for computation offloading. A nested two stage game formulation is used in [6], where the objective is to minimize both power consumption and service response time. Game theory is used in [17] [18] as a framework for designing decentralized algorithms, so that users can self-organize and make good computation offloading decisions.

We consider a set of mobile users that access cloud services over a shared base station communication channel, as shown in Figure 1. The mobile users employ computation offloading to reduce their energy usage, by uploading and executing jobs on the remote cloud servers. Time slots on the shared channel are assigned in a round-robin fashion to the set of mobile users *who decide to upload their jobs* (as opposed to those that decide to execute their job locally). Since the channel quality may be different for each user, the achievable bit rate in a given time slot may vary greatly between users. It is assumed that the arriving jobs have hard deadline completion constraints. This may restrict the set of users that can use computation offloading when job completion deadlines cannot be met.

The users compete for a common resource (the channel) while they are trying to minimize their utility (energy consumption). They act in a decentralized environment, i.e., they are allowed to make their own uploading decisions, without a central authority imposing such decisions, and according to their utility and the information about the system they can obtain from a central cloud controller/scheduler. The natural way of modelling such a setting is as a game, which the users play using the information provided by the controller, until they reach a stable state where no one would benefit by defecting, i.e., a *Nash Equilibrium (NE)*. We emphasize that although the controller controls the flow of system information from and to the users, it is unable to directly impose any uploading decisions to the users, due to the decentralized decision making setting. However, it can influence these decisions, e.g., by manipulating the information it transmits to the users; we are going to use this ability of the controller, in order to enforce a Nash equilibrium on the system by first computing a NE at the controller, and then transmitting to the users the job delays that result from the strategies in this NE. That will force all users to adopt the NE decisions (since, according to what they see as the other users' decisions, a deviation would increase energy consumption), and will stabilize the system in one round, without having to wait for the game to be played until a NE is reached.

A. Contributions

Modelling of computation offloading as a competitive game where the users try to minimize their own energy consumption has been done in previous work, and as described above, our model is closest to the model of [1] [17] and [18]. Unlike [17] [18] (which do not refer to job deadlines at all), and [1] (which does mention job deadlines, but does not include them in the formulation of their model), our model takes into account the constraints imposed by the job execution deadlines. These constraints radically change the game, since if too many users decide to offload, the per user data rate may decrease, and job completion time constraints may be violated. When this happens, users will be forced to withdraw from computation offloading. In an earlier version of this work [19], the energy for a given user was independent of other users' offload decisions. In this version however, a more sophisticated energy model couples the users' energy through their shared channel contention. As a result, the Nash equilibrium proof requires a far more sophisticated argument, and is also now given for the general user parameter case.

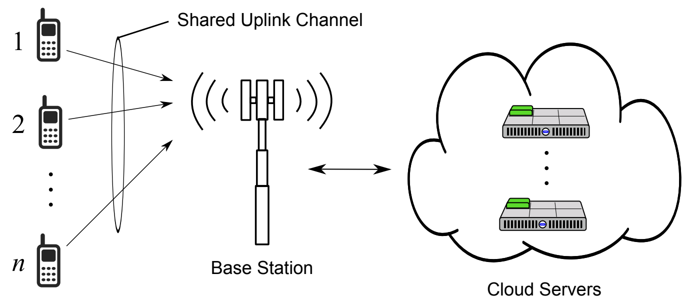

Fig. 1. Mobile Computation Offloading Model. n mobile users access infrastructure-based cloud servers over a shared wireless communication channel.

The paper shows that for a variety of parameters, a game where each user independently adjusts its offload decisions always has a pure NE, which can be explicitly described, a task that is significantly complicated by the existence of job deadline constraints. This description can be used to calculate this specific NE; alternatively, a Gauss-Seidel-like method is introduced for calculating a (possibly different) NE, and results are presented that illustrate that the system almost always converges to a NE using this method.

The fact that we are able not only to prove the existence of a NE, but to explicitly characterize and efficiently compute it, while subject to constraints imposed by the job execution deadlines, channel bit rates due to varying channel quality, and the competition over the shared channel is arguably the most significant contribution of this work. An important assumption we make in order for this approach to work is the following *truthfulness assumption*: the users report to the controller their actual decisions and parameter values, and the controller reports the actual execution times corresponding to the user decisions (either the actual or, for our case, the calculated NE ones). Enforcing this assumption is beyond the scope of this paper, and is left as an open problem for further research. Since there are many NE, data are presented which show the number of iterations needed, and the quality of the solutions obtained, by comparing the total energy consumed at the equilibrium achieved to the optimal total energy consumption, if there were a central coordinator with the ability to impose uploading decisions to the users (social cost). In particular, we find that the solutions perform well compared to a lower bound on total energy performance.

II. System Model

The system considered is shown in Figure 1. A set of n mobile users employ cloud-based computation offloading, where jobs may be executed either locally, or on remote cloud servers. If remote execution is chosen, a user must upload job-specific data that is needed to run the job on the remote server. When a set of users choose the remote execution option, they upload their job data through a communication channel that is shared among the uploading users using a round robin time slot assignment. It is assumed that the transmit power of all the users is fixed, and therefore the uploading bit rates may be different for each user due to differing radio propagation path loss values. The user data payload therefore depends on which user is currently transmitting. It is assumed that the upload bit rates are constant for the duration of a given job contention/uploading cycle [17][18][20].

We are interested in the total energy needed to execute a set of n jobs, one for each user, once the users have decided on local or remote execution during a job contention round. If a user decides to upload, it transitions its wireless air interface from a low power mode into the active state. The wireless communication channel is then shared in a round-robin fashion between those users that have made upload decisions. While a given station is participating in job uploading, its radio interface transitions between time slots during which it is actively transmitting on the uplink, and those where it is in an active waiting state where packet reception is enabled. More formal definitions are given in the following development, and Table I summarizes the notation used.

User U_m, for $m \in \{1, 2, \dots, n\}$ is characterized by the tuple $(J_m, L_m, R_m, T_m^{\max})$, which contains the following information:

- $J_m = (D_m, B_m)$, where D_m is the number of required CPU cycles in order to execute job J_m, and B_m denotes how many bits U_m needs to upload to the cloud in order to execute the job remotely.
- $L_m = (v_m^l, f_m^l)$, where v_m^l is the energy consumption per CPU cycle, and f_m^l is the number of CPU cycles executed per second if U_m decides to execute its job *locally*, i.e., without uploading it to the cloud.
- $R_m = (P_m^t, P_m^w, r_m)$, where P_m^t and P_m^w are the wireless transmission and waiting power consumption respectively, and r_m is the wireless uplink data rate for U_m.
- $T_m^{\max}$ is U_m's maximum tolerable response time.

In order to simplify our notation in the following, we also define $\beta_m = B_m/r_m$, and $\tau_m = T_m^{\max} - \frac{D_m}{f_m^s}$.

Each user U_m has a decision variable a_m that indicates whether the user decides to execute its task locally ($a_m = 0$) or upload it to the cloud ($a_m = 1$). On the cloud server side, we will use f^s to denote the server computation power. We emphasize that the server computation power is not a system bottleneck, i.e., there are always enough cloud servers to execute uploaded jobs.

The game can be imagined to be played as a sequence of iterations: During each iteration, each user U_m communicates its current decision value, a_m, to a cloud-hosted controller. The controller then provides feedback to the users, indicating the achieved response times that are attained by each. Following this, the users update their decisions and continue on until an equilibrium is reached. Once this happens, job uploading and processing occurs. In reality, the controller will collect the users' parameters and will calculate a Nash equilibrium. As mentioned in the Introduction, we assume that the users and the controller are truthful in reporting data to each other. Then the controller will communicate to the users the calculated *equilibrium delays*, so that the users will be "forced" to decide according to the equilibrium. Note that it is only the collection and dissemination of information that is centralized, not the decisions taken; the latter are taken independently by the users, and are not dictated by a central authority.

TABLE I
TABLE OF NOTATION

D_m	required CPU cycles
B_m	input bits
v_m^l	local energy consumption (joules/CPU cycle)
f_m^l	local computation power (CPU cycles/second)
f^s	cloud server computation power (CPU cycles/second)
T_m^l	local execution response time (seconds)
E_m^l	local execution energy consumption (joules)
P_m^t	transmission power (watts)
P_m^w	waiting power (watts)
r_m	channel data rate (bps)
T_m^{up}	uploading time delay (seconds)
E_m^{up}	uploading energy consumption (joules)
T_m^s	server execution time delay (seconds)
T_m^r	total remote execution response time (seconds)
E_m^r	total remote execution energy consumption (joules)
T_m	total response time (seconds)
E_m	total energy consumption (joules)
$T_m^{\max}$	maximum tolerable response time (seconds)
β_m	exclusive data uploading time (seconds)
τ_m	maximum tolerable data uploading time (seconds)
Φ_m	negative uploading time margin (seconds)
U_m	m_{th} user
n	number of users

A. Local Processing

In the case where user U_m decides to execute its job locally, we use the simple model described in [21] where the local execution energy consumption E_m^l and the time delay due to local computation T_m^l are defined as follows:

$$T_m^l = \frac{D_m}{f_m^l}, \quad E_m^l = v_m^l D_m.$$

B. Remote Processing

In the case of uploading, we describe both the wireless communication model used, and the cloud server execution model, in terms of energy consumption and time delay.

Wireless Channel Sharing: All users share a single wireless communication channel to upload their jobs. It is assumed that if m users decide to upload, time slots are shared in a round-robin fashion between them. Without loss of generality, we assume that the users are sorted so that $\beta_1 \le \beta_2 \le \cdots \le \beta_n$ and that user U_m's upload time is given by T_m^{up}. This is the equivalent of having n users with input data of $\frac{B_m}{r_m}$ bits and an uplink data rate of 1 bit per second. User m's uploading time delay is denoted by T_m^{up}. After user U_m finishes its data transmission, user U_{m+1} continues sharing the channel with the remaining users. Since users have an uplink data rate of 1 bit per second, both users U_{m+1} and U_m transmitted $\frac{B_m}{r_m}$

during the first T_m^{up} seconds. After user U_m finishes its data transmission, user U_{m+1} continues sharing the channel with the remaining users to upload the remaining data, $(\frac{B_{m-1}}{r_{m-1}} - \frac{B_m}{r_m})$. Assuming that the job upload times are large compared to the time slot duration and the link data rate does not change during the proccess of decision making and data uploading [17][18][20], it can easily be shown that

$$T_{m+1}^{\text{up}} = T_m^{\text{up}} + (\beta_{m+1} - \beta_m)\,\eta_{m+1} \tag{1}$$

where η_{m+1} is the number of users who are still uploading after user m finishes its data transmission, and $1/\eta_{m+1}$ is the normalized per user data rate. Hence $\eta_{m+1} = \sum_{i=m+1}^{n} a_i$, and, therefore, (1) implies for an uploading user U_m (i.e., $a_m = 1$), that

$$T_m^{\text{up}} = \begin{cases} (1 + \sum_{i=m+1}^{n} a_i)\beta_m & \text{if } m = 1 \\ \sum_{i=1}^{m-1} a_i\beta_i + (1 + \sum_{i=m+1}^{n} a_i)\beta_m & \text{if } 1 < m < n \\ \sum_{i=1}^{m-1} a_i\beta_i + \beta_m & \text{if } m = n \end{cases} \tag{2}$$

E_m^{up} is the energy consumption due to uploading via the wireless channel and can be calculated as transmission power times exclusive uploading time, i.e.,

$$E_m^{\text{up}} = P_m^t\beta_m + P_m^w(T_m^{\text{up}} - \beta_m) \tag{3}$$

D_m and B_m are two independent parameters. A small data upload for example, could require a large CPU execution requirement, and vice versa, i.e., the uploading bits, B_m, and the required CPU cycles, D_m, are not the same. D_m comes from the computation level of the task and both local execution time T_m^l and energy consumption E_m^l depend on D_m.

Cloud server execution: We assume that once a job has been uploaded to a cloud server, it starts executing without delay, i.e., the congestion is on the shared channel, not the cloud server. The server execution time for U_m is given by

$$T_m^s = \frac{D_m}{f^s} \tag{4}$$

We assume that the user switches to a low power sleep mode while it is waiting for its server to execute the application, as in [17], [20] and [22]. As in typical cellular and wireless LAN air interface protocols, when in sleep mode the user's radio awakens periodically, so that the user can test for packets waiting at the Base Station for downlink transmission. In this way, the users can quickly re-synchronize to the cloud server upload activity. Considering the high computation capability of the server and low sleep power, the energy consumption in this mode can be considered negligible by comparison. The total remote execution time and the total remote energy consumption are given by

$$T_m^r = T_m^{\text{up}} + T_m^s \tag{5}$$

$$E_m^r = E_m^{\text{up}} = P_m^t\beta_m + P_m^w(T_m^{\text{up}} - \beta_m) \tag{6}$$

and, by taking into account U_m's decision variable a_m, we find that its total response time and energy consumption are given by

$$T_m = a_m T_m^r + (1 - a_m)T_m^l \tag{7}$$

$$E_m = a_m E_m^r + (1 - a_m)E_m^l \tag{8}$$

Note that in this development we have assumed that other system delays, such as the communication latency between the base station and the cloud servers, are negligible compared to the others. However, these delays can be included in the formulation, if desired.

III. Central Decision Making

In conventional mobile cloud computing, a central scheduler is used to determine the decision variables a_m for all users, so that either the overall or maximum energy consumption is minimized, ensuring that all users' response time constraints are respected. Therefore, the central scheduler solves one of the following mathematical programs. In (OPT_SUM) the central scheduler minimizes the social (total) energy consumption:

$$\begin{aligned} \min_{\{a_1,a_2,\ldots,a_n\}} \quad & \sum_{m=1}^{n} E_m \quad \text{s.t.} \\ & T_m \le T_m^{\max}, \quad \forall m \in \{1,\ldots,n\} \\ & a_m \in \{0,1\}, \quad \forall m \in \{1,\ldots,n\} \end{aligned} \tag{OPT\_SUM}$$

Using (2), (6) and (8), the objective function of (OPT_SUM) can be written as

$$\begin{aligned} \sum_{m=1}^{n} E_m = \sum_{m=1}^{n} (P_m^t - P_m^w)\beta_m a_m + \sum_{m=1}^{n} (1 - a_m)v_m^l D_m \\ + \sum_{m=1}^{n} a_m P_m^w(\sum_{i<m} a_i\beta_i + \beta_m \sum_{i>m} a_i) \end{aligned} \tag{9}$$

IV. Selfish Decision Making

One of the characteristics of cloud computing is the lack of a central coordinator that can force users to upload their jobs to the cloud. Therefore, in our model we allow the mobile users to act as *selfish agents*, i.e., they decide by themselves whether to perform their computation remotely or locally, according to their own cost function. As a result, the value of a_m is set by user U_m itself; the role of the central scheduler of Section III is to just provide the agents with channel information. As a result, we adopt a game theoretic approach in order to study our setting, which requires truthfull users and controller.

In our model, each user wants to minimize its own energy consumption. The objective for a user U_m can be modeled as follows: Let $a_{-m} = (a_1, ..., a_{m-1}, a_{m+1}, ..., a_n)$ be the tuple of the offloading decisions by all other users except user U_m; then, given a_{-m}, user U_m would like to set its decision variable $a_m \in \{0, 1\}$ to the solution of the following:

$$\begin{aligned} \min_{a_m} \quad & E_m \quad \text{s.t.} \\ T_m(a_m, a_{-m}) & \le T_m^{\max} \\ a_m & \in \{0,1\} \end{aligned} \tag{mOPT}$$

Note that (7) and (8) imply that the objective and time constraint depend *on* a_m *and* a_{-m}. Therefore, (mOPT) is an optimization problem with a non-trivial solution.

Following the classic definition of Nash equilibria, suppose that there is a vector $\bar{\mathcal{A}} = (\bar{a}_1, \ldots,\ \bar{a}_n)$ such that for each

Algorithm 1 Gauss-Seidel Algorithm

```
1: procedure FindNashEquilibrium
2:     sort users so that β_1 ≤ β_2 ≤ ··· ≤ β_n
3:     randomly pick a binary vector A = (a_1, ..., a_n)
4:     N = {1, 2, ..., n}
5:     for k = 1 → n do
6:         m ← a randomly picked number from the set N.
7:         x_opt ← solution of (mOPT) for user U_m
8:         if x_opt ≠ a_m then
9:             a_m ← x_opt
10:            go to line 4
11:        else
12:            remove m from the set N.
13:    endfor
14:    return A
```

U_m, the value $\bar{a}_m$ solves (mOPT) with a_{-m} fixed to $\bar{\mathcal{A}}_{-m}$. Then $\bar{\mathcal{A}}$ is called a *(generalized) Nash equilibrium.*

In order to measure the (in)efficiency of Nash equilibria, Koutsoupias & Papadimitriou [23] introduced the notion of the *Price of Anarchy (PoA).* This is defined as the ratio of the worst-case overall (social) cost of a Nash equilibrium over the overall (centralized) optimal cost. In our experiments we do not compute necessarily the worst-case equilibrium, but we will abuse the notation by defining the 'price of anarchy' as the ratio of the cost of the reached equilibrium over the (centralized) optimal cost. We leave the estimation of PoA in the sense of [23] as an open problem.

In order to find a Nash Equilibrium (albeit not necessarily the worst-case one), we use the classic Gauss-Seidel method (Algorithm 1). In the first step we randomly choose $\mathcal{A} = (a_1, a_2, \ldots, a_n)$ where $a_i \in \{0, 1\}$ as our starting point. In most cases the starting point is not feasible (some time constraints may be violated). Then, in each iteration, user U_m is selected randomly and we solve its (mOPT) with the given $\mathbf{a}$. If the optimal solution of (mOPT) is different than the current decision value a_m, we set a_m to the new optimal solution; otherwise, we randomly select another user and continue. This iterative procedure continues until none of the user decision variables change anymore, at which point the algorithm returns the Nash equilibrium.

V. Nash Equilibrium Existence

In general, each user U_m solves (mOPT) throughout the duration of the game. If we define

$$\tau_m = T_m^{\max} - \frac{D_m}{f^s}, \tag{10}$$

then we can rewrite (mOPT) as

$$\begin{aligned} \min_{a_m} \quad & a_m P_m^w \Big(\frac{P_m^t \beta_m}{P_m^w} - \beta_m + T_m^{\text{up}}(a_{-m})\Big) + \\ & +(1 - a_m) D_m v_m^l \quad \text{s.t.} \\ & a_m T_m^{\text{up}}(a_{-m}) \le a_m \tau_m \\ & a_m \in \{0, 1\} \end{aligned} \tag{mOPT'}$$

Given Φ_m as follows

$$\Phi_m = T_m^{\text{up}} - \min\{\tau_m, \frac{v_m^l D_m}{P_m^w} - (\frac{P_m^t}{P_m^w} - 1)\frac{B_m}{r_m}\} \tag{11}$$

the optimization problem (mOPT”) would be equivalent to (mOPT')

$$\begin{aligned} \min_{a_m} \quad & a_m \Phi_m \quad \text{s.t.} \\ & a_m \in \{0, 1\} \end{aligned} \tag{mOPT”}$$

We can rewrite (mOPT”) as the following two-part definition in which Φ_m is defined by (11)

$$a_m = \begin{cases} 1 & \text{if } \Phi_m \le 0 \\ 0 & \text{if } \Phi_m > 0 \end{cases}$$

To prove the existence of Nash equilibrium in such a system we provide an algorithm (Algorithm 2) which assures convergence to a Nash equilibrium. Obviously this algorithm

Algorithm 2 Finding Nash equilibrium in hetrogeneous system

```
procedure FindNashEquilibrium(a_1, ..., a_n)
    S ← {1, ..., n}
    A = 1_{1×n}
    while max_{t∈S} Φ_t(a_t, a_{-t}) > 0 do
        k ← arg max_{t∈S} Φ_t(a_t, a_{-t})
        a_k ← 0
        remove k from set S
    return A
```

will converge in at most n iterations. We need to prove that the convergence point is a Nash equilibrium.

Theorem 1. *Algorithm 2 always converges to a Nash equilibrium.*

Proof. We claim that whenever a user leaves S, he will never be able to get back to S (i.e., to offload) during the procedure, without violating his deadline constraint. Moreover, at the end of the algorithm, no user in S has an incentive to prefer local execution instead of offloading. Together, these two claims prove the theorem.

Claim 1. *A user that has left S before iteration k ($1 \le k \le n$), will not be eligible to get back to S (i.e., to offload) right after iteration k, without violating his deadline constraint.*

Proof of Claim 1. We will prove the claim using induction on k. For $k = 1$ the claim is obviously true, since no user has left S before the current one, and the latter leaves S in iteration 1 because of his time constraint violation. We assume that it is true for all iterations $0 \le k \le m$, and we prove it for iteration $m + 1$.

Let $U_{\rho_{m+1}}$ be the player examined in iteration $m+1$ of the algorithm, and U_{ρ_i} ($1 \le i \le m$) the player removed from S in iteration i. For instance, if U_{10}, U_2, U_{12}, U_1 are removed in this order during the first four iterations, then $\rho_1 = 10$, $\rho_2 = 2$, $\rho_3 = 12$ and $\rho_4 = 1$. Due to the inductive hypothesis, none of the U_i's ($1 \le i \le m$) have entered S before iteration $m+1$. In iteration i, U_{ρ_i} ($1 \le i \le m+1$) leaves S, i.e., he changes his decision a_{ρ_i} to 0 (from 1) (i.e., from offloading to local execution instead).

Let $\vec{\mathbf{x}}_{\rho i}$ ($1 \leq i \leq m+1$) be the decision vector which indicates that U_{ρ_i} changes back to offloading ($a_{\rho_i} = 1$) after $U_{\rho_{m+1}}$ changes a_{ρ_i} to 0 (from 1):

$$\vec{\mathbf{x}}_i = (a_{\rho_i} = 1, a_{\rho_1} = \ldots = a_{\rho_{i-1}} = a_{\rho_{i+1}} = \ldots = 0, \\ a_{\rho_i} = 1, a_{-\{\rho_1,\ldots,\rho_i\}})$$

(a_{-W} indicates the decision variables for all users who are not members of set W). We want to prove that none of the U_{ρ_i}'s ($1 \leq i \leq m$) prefer to offload right after iteration $m+1$, or, equivalently,

$$\Phi_{\rho_i}(\vec{\mathbf{x}}_{\rho_i}) > 0, \ 1 \leq i \leq m \tag{12}$$

Obviously, we already have that

$$\Phi_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) > 0. \tag{13}$$

We show (12) by induction on i, starting from $i = m$ and going towards $i = 1$.

For the base case ($i = m$), we need to show that after the departure of $U_{\rho_{m+1}}$ in iteration $m+1$, U_{ρ_m} cannot return to S, or, equivalently, that $\Phi_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) > 0$. Since player U_{ρ_m} left S at iteration m, we have (due to line 7 in the algorithm)

$$\Phi_{\rho_m}(\vec{\mathbf{a}}) > \Phi_{\rho_{m+1}}(\vec{\mathbf{a}}) \tag{14}$$

where we define

$$\vec{\mathbf{a}} = \{a_{\rho_m}, a_{\rho_{m+1}} = 1, a_{-\{\rho_m,\rho_{m+1}\}}\}.$$

There are two possible cases:

Case 1 - 1. $\rho_m > \rho_{m+1}$

Equation (2) shows that

$$T^{\text{up}}_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) = T^{\text{up}}_{\rho_m}(\vec{\mathbf{a}}) - \beta_{\rho_m} \\ \Rightarrow \Phi_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) = \Phi_{\rho_m}(\vec{\mathbf{a}}) - \beta_{\rho_m} \tag{15}$$

$$T^{\text{up}}_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) = T^{\text{up}}_{\rho_{m+1}}(\vec{\mathbf{a}}) - \beta_{\rho_m} \\ \Rightarrow \Phi_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) = \Phi_{\rho_{m+1}}(\vec{\mathbf{a}}) - \beta_{\rho_m} \tag{16}$$

Then equations (14), (13), (15), and (16) show that

$$\Phi_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) > \Phi_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) > 0.$$

Case 1 - 2. $\rho_m < \rho_{m+1}$

According to equation (2) we would have

$$T^{\text{up}}_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) = T^{\text{up}}_{\rho_m}(\vec{\mathbf{a}}) - \beta_{\rho_m} \\ \Rightarrow \Phi_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) = \Phi_{\rho_m}(\vec{\mathbf{a}}) - \beta_{\rho_{m+1}} \tag{17}$$

$$T^{\text{up}}_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) = T^{\text{up}}_{\rho_{m+1}}(\vec{\mathbf{a}}) - \beta_{\rho_m} \\ \Rightarrow \Phi_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) = \Phi_{\rho_{m+1}}(\vec{\mathbf{a}}) - \beta_{\rho_{m+1}} \tag{18}$$

Then equations (14), (13), (17), and 18 show that

$$\Phi_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) > \Phi_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) > 0.$$

Hence U_{ρ_m} cannot offload right after $U_{\rho_{m+1}}$ decides to execute locally in iteration $m+1$. We will assume that (12) is true for all $l \leq i \leq m$, and we prove it for $i = l-1$. There are two possible cases:

Case 2 - 1. $\rho_{l-1} < \rho_{m+1}$

If we consider indices $\{\rho_{l-1}, \ldots, \rho_{m+1}\}$ in ascending order, then let ρ_z be the index right after ρ_{l-1}, and $\rho_{\min}, \rho_{\max}$ be the smallest and biggest index respectively, i.e.,

$$\rho_{\min} < \cdots < \rho_{l-1} < \rho_z < \cdots < \rho_{\max}$$

Note that for the case we are considering, indices $\rho_z, \rho_{\min}, \rho_{\max}$ are well defined, even if the first one may coincide with the third. We also define the sets S_L and S_U as follows:

$$S_L = \{\rho_j : l-1 < j \leq m+1, j \neq z, \rho_j < \rho_{l-1}\} \\ S_U = \{\rho_j : l-1 < j \leq m+1, j \neq z, \rho_j > \rho_z\}$$

Since $U_{\rho_{l-1}}$ was removed from S before U_{ρ_z}, we have

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{a}}) \geq \Phi_{\rho_z}(\vec{\mathbf{a}}) \tag{19}$$

where we define

$$\vec{\mathbf{a}} = \{a_{\{\rho_{l-1},\rho_z\}\cup S_L \cup S_U} = 1, a_{-\{\rho_{l-1},\rho_z\}\cup S_L \cup S_U}\}.$$

Equation (2) implies that

$$T^{\text{up}}_{\rho_{l-1}}(\vec{\mathbf{b}}) = T^{\text{up}}_{\rho_{l-1}}(\vec{\mathbf{a}}) - \sum_{j \in S_L} \beta_j \Rightarrow \\ \Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) = \Phi_{\rho_{l-1}}(\vec{\mathbf{a}}) - \sum_{j \in S_L} \beta_j \tag{20}$$

$$T^{\text{up}}_{\rho_z}(\vec{\mathbf{b}}) = T^{\text{up}}_{\rho_z}(\vec{\mathbf{a}}) - \sum_{j \in S_L} \beta_j \Rightarrow \\ \Phi_{\rho_z}(\vec{\mathbf{b}}) = \Phi_{\rho_z}(\vec{\mathbf{a}}) - \sum_{j \in S_L} \beta_j \tag{21}$$

where we define

$$\vec{\mathbf{b}} = \{a_{S_L} = 0, a_{\{\rho_{l-1},\rho_z\}\cup S_U} = 1, a_{-\{\rho_{l-1},\rho_z\}\cup S_L \cup S_U}\}.$$

Equations (19), (20), and (21) show that

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) \geq \Phi_{\rho_z}(\vec{\mathbf{b}}) \tag{22}$$

Equation (2) also implies that

$$T^{\text{up}}_{\rho_{l-1}}(\vec{\mathbf{c}}) = T^{\text{up}}_{\rho_{l-1}}(\vec{\mathbf{b}}) - |S_U|\beta_{\rho_{l-1}} \Rightarrow \\ \Phi_{\rho_{l-1}}(\vec{\mathbf{c}}) = \Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) - |S_U|\beta_{\rho_{l-1}} \tag{23}$$

$$T^{\text{up}}_{\rho_z}(\vec{\mathbf{c}}) = T^{\text{up}}_{\rho_z}(\vec{\mathbf{b}}) - |S_U|\beta_{\rho_z} \Rightarrow \\ \Phi_{\rho_z}(\vec{\mathbf{c}}) = \Phi_{\rho_z}(\vec{\mathbf{b}}) - |S_U|\beta_{\rho_z} \tag{24}$$

where we define

$$\vec{\mathbf{c}} = \{a_{S_L \cup S_U} = 0, a_{\{\rho_{l-1},\rho_z\}} = 1, a_{-\{\rho_{l-1},\rho_z\}\cup S_L \cup S_U}\}.$$

The fact that $\beta_{\rho_{l-1}} \leq \beta_{\rho_z}$, together with equations (22), (23), and (24), implies that

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{c}}) \geq \Phi_{\rho_z}(\vec{\mathbf{c}}) \tag{25}$$

Then (2) implies that

$$T^{\text{up}}_{\rho_{l-1}}(\vec{\mathbf{x}}_{\rho_{l-1}}) = T^{\text{up}}_{\rho_{l-1}}(\vec{\mathbf{c}}) - \beta_{\rho_{l-1}} \Rightarrow \\ \Phi_{\rho_{l-1}}(\vec{\mathbf{x}}_{\rho_{l-1}}) = \Phi_{\rho_{l-1}}(\vec{\mathbf{c}}) - \beta_{\rho_{l-1}} \tag{26}$$

$$T^{\text{up}}_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) = T^{\text{up}}_{\rho_z}(\vec{\mathbf{c}}) - \beta_{\rho_{l-1}} \Rightarrow \\ \Phi_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) = \Phi_{\rho_z}(\vec{\mathbf{c}}) - \beta_{\rho_{l-1}} \tag{27}$$

and, therfore, from the inductive hypothesis and equations (25), (26), and (27), we get that

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{x}}_{\rho_{l-1}}) \geq \Phi_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) \geq 0.$$

Case 2 - 2. $\rho_{l-1} > \rho_{m+1}$

This case can be divided into two subcases.

*a) **Subcase 1**:* There exists a $l-1 < z \leq m+1$ such that $\rho_z > \rho_{l-1}$. In this case we can use exactly the same arguments as in the previous case.

*b) **Subcase 2**:* For all $l-1 < y \leq m+1$ we have that $\rho_y < \rho_{l-1}$. If we consider indices $\{\rho_{l-1}, \ldots, \rho_{m+1}\}$ in ascending order, then let ρ_z be the index right before ρ_{l-1}. Again, we define S_L and S_U as follows:

$$S_L = \{\rho_j : l-1 < j \leq m+1, j \neq z, \rho_j < \rho_z\}$$
$$S_U = \{\rho_j : l-1 < j \leq m+1, j \neq z, \rho_j > \rho_{l-1}\}$$

Obviously by this definition we have $S_U = \emptyset$ and $S_L = \{\rho_j : j \neq z, l-1 < j \leq m+1\}$. Since $U_{\rho_{l-1}}$ was removed before U_{ρ_z}, we have

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{a}}) \geq \Phi_{\rho_z}(\vec{\mathbf{a}}) \tag{28}$$

where we define

$$\vec{\mathbf{a}} = \{a_{\{\rho_{l-1},\rho_z\}\cup S_L} = 1, a_{-\{\rho_{l-1},\rho_z\}\cup S_L}\}.$$

Equation (2) implies the same equations as (20) and (21) (recall that $S_U = \emptyset$ in $\vec{\mathbf{b}}$), and therefore we get that

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) \geq \Phi_{\rho_z}(\vec{\mathbf{b}}). \tag{29}$$

We also have

$$T^{\text{up}}_{\rho_{l-1}}(\vec{\mathbf{x}}_{l-1}) = T^{\text{up}}_{\rho_{l-1}}(\vec{\mathbf{b}}) - \beta_{\rho_z} \Rightarrow$$
$$\Phi_{\rho_{l-1}}(\vec{\mathbf{x}}_{l-1}) = \Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) - \beta_{\rho_z} \tag{30}$$
$$T^{\text{up}}_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) = T^{\text{up}}_{\rho_z}(\vec{\mathbf{b}}) - \beta_{\rho_z} \Rightarrow$$
$$\Phi_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) = \Phi_{\rho_z}(\vec{\mathbf{b}}) - \beta_{\rho_z} \tag{31}$$

Equations (29), (30), and (31) indicate that

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{x}}_{\rho_{l-1}}) \geq \Phi_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) > 0.$$

□

Claim 2. *At the end of algorithm 2, all users in S will offload.*

Proof (of Claim 2). The algorithm terminates either with $S = \emptyset$ or because $\Phi_j \leq 0$ in line 9. In the latter case, we have $\Phi_i \leq \Phi_j \leq 0$ for all $i \in S$, and, therefore, no user in S has an incentive to not offload. □

Claims 1 and 2 together prove the theorem. □

VI. Experimental Results

The Monte Carlo method was used to evaluate the efficiency of the game theoretic model, the convergence time and the energy consumption attained at the Nash equilibrium points (NEPs). In order to cover a wide range of scenarios, 500 random configurations were generated and each was executed 500 times with different starting decision values and random seeds. In all configurations, parameters were generated using a random uniform distribution. The required CPU cycles, D_m, were chosen randomly between 2.5 and 25 Gcycles. Input data size, B_m, is between 0.42 to 42 Mb and the channel data rate, r_m, ranged from 6.4 to 64 Mbps. Local computation power, f^l_m, was selected randomly from 0.5, 0.8 or 1 giga CPU cycles/sec and cloud server computation power, f^s, was taken to be 100 giga CPU cycles/sec. Data transmission power, P^t_m, was between 0.75 to 1 mW [24][25]. Local energy consumption, v^l_m, is considered to be equal to P^l_m/f^l_m and local execution power consumption, P^l_m, was chosen randomly from 8 , 9 and 10 mW [24][26][27][28].

Figure 2 shows the number of iterations required to converge to a Nash equilibrium point. The two dashed lines show the maximum number of iterations among all reached NEPs for Algorithm 2 and the Gauss-Seidel algorithm. Algorithm 2 shows a better performance since it converges in at most n iterations while the Gauss-Seidel algorithm convergence time is random and could potentially loop forever. This behaviour however, was not observed in any of our experiments, i.e., Gauss-Seidel always found a Nash equilibrium. The convergence time of Algorithm 2 and the Gauss-Seidel algorithm was studied and the results are shown in Figure 3. Algorithm 2 has order of n^2 time complexity. The average convergence time of these two algorithms is shown by solid lines. In Algorithm 2, since in each iteration the maximum value of Φ needs to be calculated, increasing the number of users will increase the execution time. In addition, due to the constant channel capacity, a smaller portion of users chooses to offload in larger groups, thus Algorithm 2 needs to iterate more to converge. As a result, the average execution time of Algorithm 2 in large groups (in our simulation results, with more than 150 users) is longer than in the Gauss-Seidel method. However, simulation results show that the game theoretic computation offloading mechanism scales well with the size of the problem. The social optimum problem is NP hard and very time consuming to solve.

Table II illustrates the average offloading ratio, which is defined as the ratio of the number of remote executions to the number of users (n). As the number of users increases, proportionally fewer users offload at equilibrium. This is expected since the channel capacity is kept constant, and, therefore, the remote execution delay becomes prohibitively large for an ever greater proportion of users. Consequently, the MCC approach is more beneficial in small to moderate size groups.

In Figure 4, three different task execution approaches were studied. In the upper curve, all users execute their tasks locally while in the two lower curves, Algorithm 2 and the Gauss-Seidel algorithm were used to assign remote execution to some

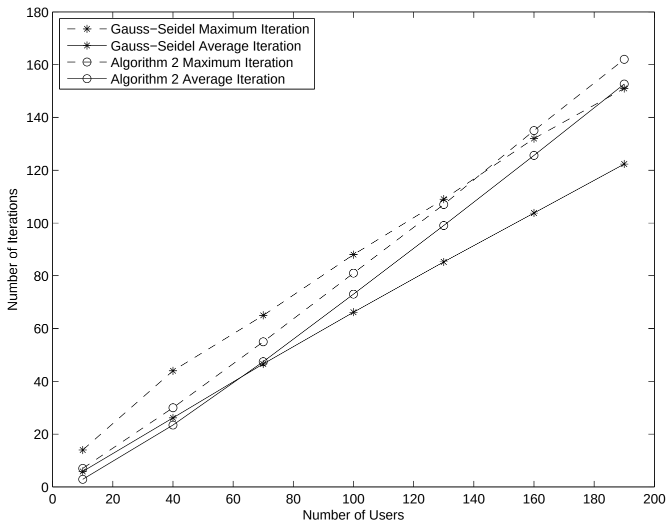

Fig. 2. Number of Iterations vs. Number of Users

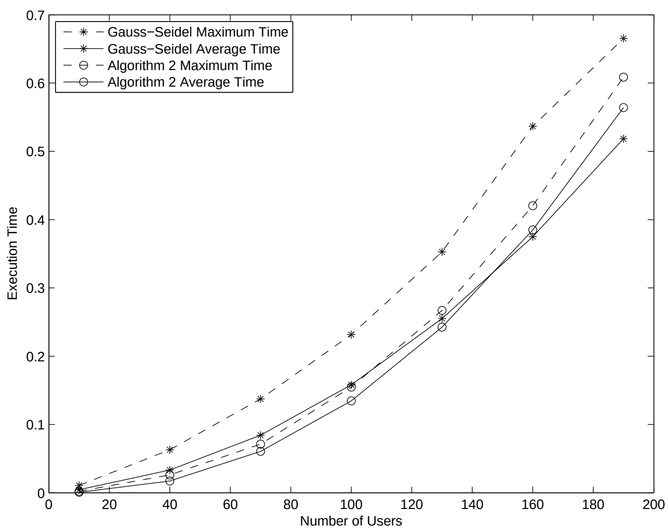

Fig. 3. Convergence Time vs. Number of Users

TABLE II
AVERAGE OFFLOADING RATIO

n	Average Offloading Ratio	n	Average Offloading Ratio
10	0.7084	110	0.2602
20	0.5553	120	0.2470
30	0.4717	130	0.2392
40	0.4140	140	0.2313
50	0.3772	150	0.2215
60	0.3438	160	0.2145
70	0.3221	170	0.2091
80	0.3025	180	0.2024
90	0.2871	190	0.1977
100	0.2731	200	0.1922

users. All social energy cost values were normalized according to the optimal social cost (OPT_SUM). The game theoretic approaches resulted in considerable energy savings. The energy cost difference between these approaches decreases by increasing the number of users since the offloading ratio becomes smaller.

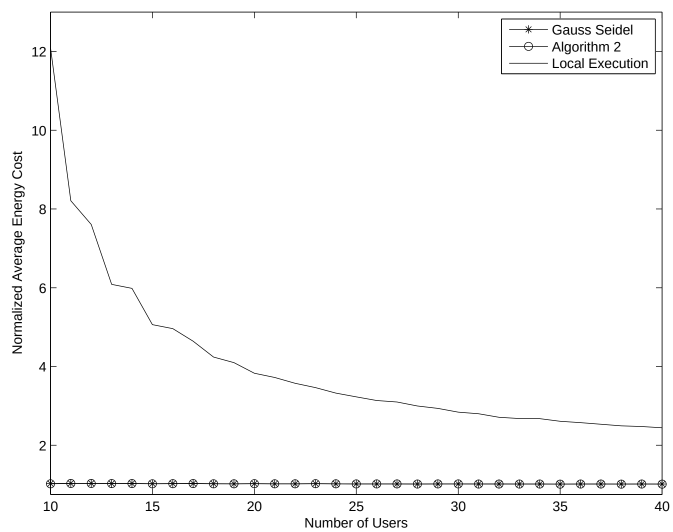

Fig. 4. Normalized Energy Cost versus Number of Users for Algorithm 2, Gauss-Seidel Algorithm and Local Execution

Figure 5 shows the ratio of the total cost at equilibrium determined by Gauss-Seidel and Algorithm 2 over the optimal social cost (OPT_SUM). More specifically, we show the ratio for the worst (total cost-wise) equilibria reached (WE/SO) and the average over all reached equilibria (AE/SO). While the cost for the worst equilibrium may be as much as 300% higher than the social optimum, the average cost of a reached equilibrium is much closer to the social optimum. Therefore the lack of central coordination to solve (OPT_SUM) does not result in a prohibitive increase in the total energy needed for supporting offloading. Since the number of Nash equilibrium points increases in larger groups, the probability of hitting the worst equilibrium will decrease. As a result, in our experiments, by increasing the number of users, the social cost of the worst equilibrium reached was closer to the average among all reached equilibria. We leave open the question of theoretical upper bounds for the worst-case equilibrium ratio (i.e., the PoA as defined by [23]).

VII. CONCLUSIONS

In this paper we considered a system where mobile users use computation offloading, where energy consumption is reduced by executing jobs on a remote cloud server, rather than locally. In order to perform remote execution, a mobile user uploads the job over a base station channel that is shared by all of the uploading users. The jobs are subject to hard deadline constraints, and because the channel quality may be different for each user, this may restrict the ability to reduce energy usage. The system was modelled as a competitive game where users are interested in minimizing their own energy use. The paper showed that for known classes of parameters, a game where each user independently adjusts its offload decisions always has a pure Nash equilibrium. Results were presented that illustrate that the system always converges to a Nash equilibrium using the Gauss-Seidel method. Data were also presented that shows the number of iterations required, and

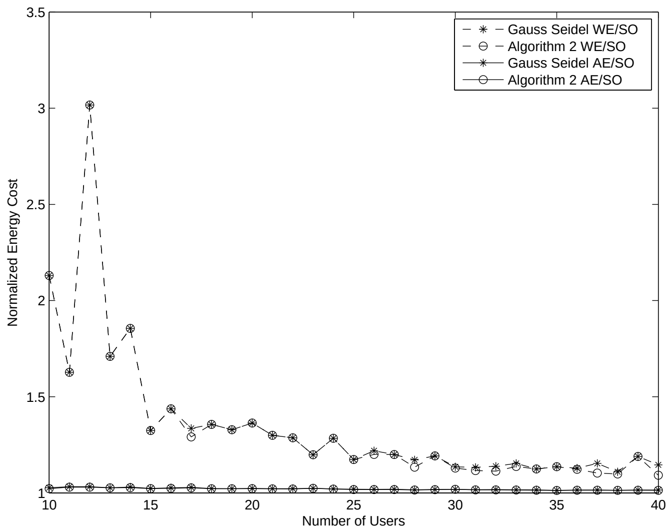

Fig. 5. Normalized Social Energy Consumption Over All Discovered NEs

the quality of the equilibria obtained. In particular, we found that the solutions perform well compared to a lower bound on total energy performance.

REFERENCES

[1] Y. Ge, Y. Zhang, Q. Qiu, and Y.-H. Lu, "A Game Theoretic Resource Allocation for Overall Energy Minimization in Mobile Cloud Computing System," in *ISLPED*, ser. ISLPED '12. ACM, July 2012, pp. 279–284.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.

[3] [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.

[4] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, "Gearing Resource-Poor Mobile Devices with Powerful Clouds: Architectures, Challenges, and Applications," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 14–22, Jun. 2013.

[5] L. Lei, Z. Zhong, K. Zheng, J. Chen, and H. Meng, "Challenges on Wireless Heterogeneous Networks for Mobile Cloud Computing," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 34–44, Jun. 2013.

[6] Y. Wang, X. Lin, and M. Pedram, "A Nested Two Stage Game-Based Optimization Framework in Mobile Cloud Computing System," in *7th International Symposium on Service Oriented System Engineering (SOSE)*. IEEE, March 2013, pp. 494–502.

[7] M. Ali, "Green Cloud on the Horizon," in *Cloud Computing*, ser. Lecture Notes in Computer Science, M. Jaatun, G. Zhao, and C. Rong, Eds. Springer Berlin Heidelberg, 2009, vol. 5931, pp. 451–459. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10665-1_41

[8] G. H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," *Computer*, vol. 27, no. 4, pp. 38–47, 1994.

[9] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving Portable Computer Battery Power through Remote Process Execution," *Journal of ACM SIGMOBILE on Mobile Computing and Communications Review*, vol. 2, no. 1, pp. 19–26, 1998.

[10] 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," *IEEE INFOCOM*, pp. 945–953, March 2012.

[11] 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 *The 8th International Conference on Mobile Systems, Applications, and Services*. ACM, June 2010, pp. 49–62.

[12] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic Execution between Mobile Device and Cloud," in *The sixth conference on Computer systems*. ACM, April 2011, pp. 301–314.

[13] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[14] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The Case for Cyber Foraging," in *The 10th Workshop on ACM SIGOPS European Workshop*, July 2002, pp. 87–92.

[15] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Communications*, vol. 8, no. 4, pp. 10–17, 2001.

[16] D. Niyato, P. Wang, E. Hossain, W. Saad, and Z. Han, "Game Theoretic Modeling of Cooperation among Service Providers in Mobile Cloud Computing Environments," in *Wireless Communications and Networking Conference (WCNC)*. IEEE, April 2012, pp. 3128–3133.

[17] X. Chen, "Decentralized Computation Offloading Game for Mobile Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, April 2014.

[18] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *arXiv preprint arXiv:1510.00888*, 2015.

[19] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy efficient offloading for competing users on a shared communication channel," in *2015 IEEE International Conference on Communications, ICC 2015, London, United Kingdom, June 8-12, 2015*, 2015, pp. 3192–3197.

[20] K. Kumar and Y. H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" *Computer*, vol. 43, no. 4, pp. 51–56, Apr. 2010.

[21] K. Kumar and Y.-H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" *IEEE Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[22] C. Xian, Y.-H. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *Parallel and Distributed Systems, 2007 International Conference on*, vol. 2. IEEE, 2007, pp. 1–8.

[23] E. Koutsoupias and C. Papadimitriou, "Worst-case Equilibria," in *16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, March 1999, pp. 404–413.

[24] M. Nir, A. Matrawy, and M. St-Hilaire, "An energy optimizing scheduler for mobile cloud computing environments," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE, 2014, pp. 404–409.

[25] J. Song, Y. Cui, M. Li, J. Qiu, and R. Buyya, "Energy-traffic tradeoff cooperative offloading for mobile cloud computing," in *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of*. IEEE, 2014, pp. 284–289.

[26] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *Wireless Communications, IEEE Transactions on*, vol. 11, no. 6, pp. 1991–1995, 2012.

[27] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Energy and performance-aware task scheduling in a mobile cloud computing environment," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 192–199.

[28] Y. Wang, X. Lin, and M. Pedram, "A nested two stage game-based optimization framework in mobile cloud computing system," in *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*. IEEE, 2013, pp. 494–502.

Erfan Meskar received the B.Sc. degree in Electrical Engineering from Amirkabir University of Technology, Tehran, Iran, in 2013, and the M.A.Sc. degree in Electrical and Computer Engineering from McMaster University, Hamilton, Ontario, Canada, in 2015. He is currently working towards the Ph.D. degree in the Department of Electrical and Computer Engineering at the University of Toronto, Toronto, Ontario, Canada. His research interests are in the area of mobile cloud computing and game theory. He has also served as a reviewer for several IEEE journals and major conferences.

George Karakostas is an associate professor in the Department of Computing & Software of McMaster University, in Hamilton ON, Canada. He holds an Eng. Diploma in Computer Engineering & Informatics from the Univercity of Patras, Greece, and MASc and PhD degrees in Computer Science from Princeton University. His main research interests are in the field of Theoretical Computer Science, and, more specifically, in the design and analysis of approximation algorithms, algorithmic game theory, and their applications in practical fields such as energy consumption and networks.

Terence D. Todd received the B.A.Sc, M.A.Sc and Ph.D degrees in Electrical Engineering from the University of Waterloo, Waterloo, Ontario, Canada. He is currently a professor of electrical and computer engineering at McMaster University in Hamilton, Ontario, Canada.

Dr. Todd spent 1991 on research leave in the Distributed Systems Research Department at AT&T Bell Laboratories in Murray Hill, NJ. During that time he worked on the characterization of one of the first ATM switches and developed techniques for mesh network media access control. Professor Todd also spent 1998 on research leave at The Olivetti and Oracle Research Laboratory in Cambridge, England. While at ORL he worked on the Piconet Project which was an early embedded wireless network testbed. Professor Todd's research interests include local area networks, wireless communications, and the performance analysis of computer communication networks. His current work includes the design of energy efficient wireless infrastructure, such as that which operates from energy sustainable sources such as solar power.

Professor Todd is a past chairholder of the NSERC/RIM/CITO Chair on Wireless Networking and is a former editor for the IEEE Transactions on Networking and the IEEE Transactions on Mobile Computing. Dr. Todd is a Professional Engineer in the province of Ontario and a member of the IEEE.

Dongmei Zhao received a Ph.D degree from the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada in June 2002. In July 2002 she joined the Department of Electrical and Computer Engineering at McMaster University, where she is a Professor. From April 2004 to March 2009 she was an adjunct assistant professor in the Department of Electrical and Computer Engineering at University of Waterloo. She has been an associate editor of the IEEE Transactions on Vehicular Technology since 2007. She also served as an editor for EURASIP Journal on Wireless Communications and Networking and Journal of Communications and Networks. She is a co-chair of the Wireless Networking Symposium in IEEE GLOBECOM Conference 2007, a co-chair of the General Symposium of the International Wireless Communications and Mobile Computing (IWCMC) Conference 2007, a co-chair of the Vehicular Networks Symposium of IWCMC from 2012 to 2016. She has been in Technical Program Committee of many international conferences in her fields. Dongmei Zhao is a member of IEEE and a Professional Engineer of Ontario. Dr. Zhao's research areas are in radio resource management and mobile cloud computing.