Energy Efficient 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*—In this paper we consider mobile users that employ *computation offloading*. In computational offloading, users can reduce energy consumption by executing jobs on a remote cloud server, rather than locally. In order to execute a job in the cloud, a mobile user must upload the job over a base station channel which is shared by all of the uploading users. The jobs are subject to hard deadline constraints, and since the channel quality may be different for each user, this may restrict the users ability to reduce energy usage. The system is modelled as a competitive game where each user is interested in minimizing its own energy use. The game is subject to the real-time constraints imposed by job execution deadlines, user specific channel bit rates, and the competition over the shared communication channel. The paper shows that for known classes of parameters, a game where each user independently adjusts its offload decisions always has a pure Nash equilibrium, and a Gauss-Seidel-like method for determining this equilibrium is presented. Results are then presented which illustrate that the system always converges to a Nash equilibrium using Gauss-Seidel. Data is presented which show the number of Nash equilibria that are found, the number of iterations required, and the quality of the solutions obtained. In particular, we find that the solutions perform well compared to a lower bound on total energy performance.**

I. INTRODUCTION

Mobile cloud computing (MCC) has rapidly evolved and has already started to revolutionize mobile device operation. MCC can improve application performance and reduce mobile user energy requirements by migrating computational tasks and data storage functions away from the user, and onto infrastructure-based cloud servers. This enables mobile users to benefit from computationally intensive applications, which would otherwise tax the resources of the user if they were run locally [1]. These capabilities have been enabled, in part, by virtualization techniques that permit cloud-based servers to run applications on behalf of their mobile counterparts [2]. According to a recent study by Cisco Inc., mobile cloud traffic is expected to increase by a factor of twelve over the next five years, with a compound annual growth rate of over 60 percent. Cloud based application support is expected to account for 90 percent of total mobile data traffic by 2018 [3].

In this paper we consider computation offloading, where mobile battery life is improved by offloading job execution to remote cloud servers, rather than performing the computations locally. The work in reference [4] has shown that remote application execution can significantly reduce mobile user energy usage in these types of situations. Computation offloading benefits from the use of fast infrastructure-based cloud servers with significantly more resources than that of a typical mobile user. There is also work which has looked at computation offloading from an application viewpoint, where jobs to be executed can be partitioned into multiple local and remote execution components [5]. In this case, algorithms must determine the best partition to be selected and executed remotely.

Reference [6] proposes an architecture referred to as MAUI, which dynamically controls computation offloading for run-time .NET applications. MAUI utilizes .NET features to profile the application and formulates the offloading problem as a linear program (LP). The work in [7] proposes a similar architecture for Android applications. Recent research has also proposed a variety of application offloading mechanisms [7][8][9][10]. Various cloud-assisted mobile platforms have also been proposed, such as cloudlet servers [8] and cloud clones [7]. In the latter case, each mobile user is associated with a system-level cloud-hosted clone which runs in a virtual machine, executing applications on behalf of the mobile user.

There have been some studies on the use of game theory in mobile cloud computing. In [11], a scenario was considered where multiple service providers cooperatively offer mobile services, and game theory was used to share revenue. Reference [1] aims to reduce energy consumption in MCC on both the server and user side, to achieve sustainability. A congestion-based game and optimization framework is used, where each mobile user is a player and the strategy is to select servers for computation offloading. A nested two stage game formulation for MCC is provided in [12], in which the objective of each mobile user is to minimize both power consumption and service response time. In [13], game theory is used as a framework for designing decentralized algorithms, so that users can self-organize and make good computation offloading decisions.

In this paper we consider a set of mobile users which access infrastructure-based cloud servers over a wireless communication channel. An example of the system is shown in Figure 1. The mobiles can choose to use computation offloading to reduce their energy use, by uploading and executing jobs on the remote cloud servers. The users share a base station

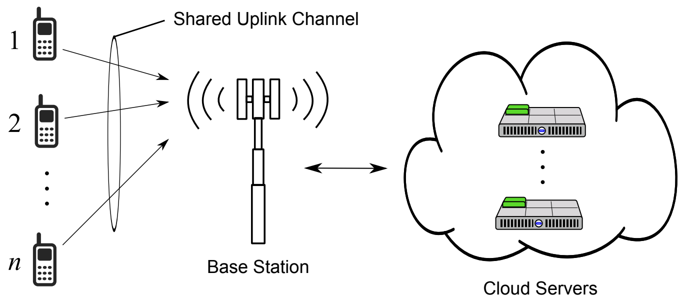

Fig. 1. Mobile Computation Offloading Model. n mobile wireless users access infrastructure-based cloud servers over a shared communication channel.

communication channel which is used for job uploading and where time slots are dynamically allocated among all of the users currently uploading. The channel quality may be different for each user however, and the achievable bit rate in a given time slot may vary greatly between users. Since the arriving jobs have hard deadline completion constraints, this may restrict the use of computation offloading when job completion deadlines cannot be met.

The system is modelled as a competitive game where the users attempt to minimize their own energy use. This activity is subject to the real-time constraints imposed by the job execution deadlines, different bit rates due to varying channel quality, and the competition over the shared communication channel. If too many users choose to offload, for example, the per user data rate may decrease to the point that job execution time constraints are violated. This may force users to withdraw from computation offloading. The paper shows that for certain classes of parameters, a game where each user independently adjusts its offload decisions always has a pure Nash equilibrium. A Gauss-Seidel-like method is presented for determining this equilibrium and results are presented which illustrate that the system always converges to a Nash equilibrium using the Gauss-Seidel method. Data is presented which show the number of equilibria that are found, the number of iterations required, and the quality of the solutions obtained. In particular, we find that the solutions perform well compared to a lower bound on total energy performance.

II. System Model

The system is shown in Figure 1 where n mobile users use cloud-based computation offloading through a shared communication channel. Table I summarizes the notation used in following development.

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.
- $R_m = (P_m^t, r_m)$, where P_m^t is the wireless transmission power consumption, and r_m is the wireless uplink data rate for U_m if it can use the channel exclusively.
- $T_m^{\max}$ is U_m's maximum tolerable response time.

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)
r_m	channel data rate (bps)
T_m^{off}	offloading time delay (seconds)
E_m^{off}	offloading 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

In order to simplify our notation in the following, we also define $\beta_m = B_m/r_m$, and $\Phi_m = T_m^{\text{off}} - \tau_m$.

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 which 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 simulate the game itself; when the simulation ends at equilibrium, it will communicate to the users the calculated *equilibrium delays*, so that the users will be 'forced' to decide according to the equilibrium.

A. Local Processing

In the case where user U_m decides to execute its job locally, we use the simple model described in [14] where the local execution energy consumption E_m^l and the time delay due to local computation T^l_m are defined as follows:

$$T^l_m = \frac{D_m}{f^l_m}, \quad E^l_m = v^l_m D_m.$$

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 $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_n$ and that user U_m's upload time is given by T^{off}_m. After user U_m finishes its data transmission, user U_{m+1} continues sharing the channel with the remaining users. Assuming that the job upload times are large compared to the time slot duration, it can easily be shown that

$$T^{\text{off}}_{m+1} = T^{\text{off}}_m + (\beta_{m+1} - \beta_m)\,\eta_{m+1} \tag{1}$$

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

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

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

$$E^{\text{off}}_m = P^t_m \beta_m \tag{3}$$

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

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

Then the total remote execution time and the total remote energy consumption are given by

$$T^r_m = T^{\text{off}}_m + T^s_m \tag{5}$$

$$E^r_m = E^{\text{off}}_m = P^t_m \beta_m \tag{6}$$

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

$$T_m = a_m T^r_m + (1 - a_m)T^l_m \tag{7}$$

$$E_m = a_m E^r_m + (1 - a_m)E^l_m \tag{8}$$

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, is 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 the overall energy consumption is minimized, ensuring that all users' response time constraints are respected. Therefore, the central scheduler solves the following mathematical program (OPT).

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

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

$$\sum_{m=1}^{n} E_m = \sum_{m=1}^{n} P^t_m \beta_m a_m + \sum_{m=1}^{n} (1 - a_m) v^l_m D^l_m \tag{9}$$

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.

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 $a_{-m} = (a_1, ..., a_{m1}, a_{m+1}, ..., a_n)$ 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:

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

Note that (8) implies that the objective depends *only on* a_m, and, in fact, the optimal decision would be $a_m = 1$, since, in this case, there is no local execution energy consumption incurred by U_m. But this may not be possible since the time constraints may be violated. Therefore, (mOPT) is an optimization problem with a non-trivial solution.

Following the classic definition of Nash equilibria, suppose that there is a vector $\bar{\mathbf{a}} = (\bar{a}_1, \ldots, \bar{a}_n)$ such that for each U_m, the value $\bar{a}_m$ solves (mOPT) with a_{-m} fixed to $\bar{a}_{-m}$. Then $\bar{\mathbf{a}}$ is called a *(generalized) Nash equilibrium.*

In order to measure the (in)efficiency of Nash equilibria, Koutsoupias & Papadimitriou [15] introduced the notion of

Algorithm 1 Gauss-Seidel Algorithm

```
 1: procedure FindNashEquilibrium
 2:     sort users so that $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_n$
 3:     randomly pick a binary vector $\mathbf{a} = (a_1, \ldots, a_n)$
 4:     NE = FALSE
 5:     $S = \emptyset$
 6:     for $m = 1 \to n$ do
 7:         if $a_m = 1$ then
 8:             add $m$ to the set $S$
 9:     endfor
10:     while NE = FALSE do
11:         $N = \{1, 2, \ldots, n\}$
12:         for $k = 1 \to n$ do
13:             $m \leftarrow$ a randomly picked number from the set $N$.
14:             $x_{\text{opt}} \leftarrow$ solution of (mOPT) for user $U_m$
15:             if $x_{\text{opt}} \neq a_m$ then
16:                 $a_m \leftarrow x_{\text{opt}}$
17:                 if $x_{\text{opt}} = 0$ then
18:                     remove $m$ from the set $S$
19:                 else
20:                     add $m$ to the set $S$
21:                 go to line 27
22:             else
23:                 remove $i$ from the set $N$.
24:                 if $|N| = 0$ then
25:                     NE = TRUE
26:                     return $\mathbf{a}$
27:         endfor
28:     endwhile
```

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 [15] 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 $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ 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 to 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 this section we prove the existence of Nash equilibria for the special cases of *homogeneous* and *semi-homogeneous* systems.

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

$$\tau_m = T_m^{\max} - \frac{D_m}{f_m^l}, \tag{10}$$

then we can rewrite (mOPT) as

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

Definition 1. *An MCC system is* semi-homogeneous *iff*

$$\tau_i = \tau_j, \quad \forall i, j \in \{1, \ldots, n\}.$$

We start Algorithm 1 with initial decisions $a_m = 0, \forall m$. In each iteration, the user with the next smallest β value is selected to solve problem (mOPT'), i.e., the users are examined in increasing order of β values. We will show that if we force the algorithm to terminate when it either encounters a user who prefers not to offload due to its time constraint, or all users decide to offload, then it has reached a Nash equilibrium.

Suppose that user U_{k+1} is the next one to be examined by the algorithm in iteration $k + 1$. Since the algorithm has not terminated yet, we have $S^k = \{1, 2, \ldots, k\}$ and

$$T_{k+1}^{\text{off}} = \sum_{i=1}^{k} a_i\beta_i + (1 + \sum_{i=k+2}^{n} a_i)\beta_{k+1} = \sum_{i=1}^{k+1} \beta_i \tag{11}$$

Theorem 1. *If user U_{k+1} decides to offload in iteration $k+1$ and $j \in S^k$, then $j \in S^{k+1}$, for all $j < k + 1$.*

Proof: Under the conditions of the theorem, we have

$$T_j^{\text{off}} = \sum_{i=1}^{j-1} a_i\beta_i + (1 + \sum_{i=j+1}^{n} a_i)\beta_j = \sum_{i=1}^{j-1} \beta_i + (k - j + 2)\beta_j$$

and, by using (11) and semi-homogeneity, we get that

$$T_{k+1}^{\text{off}} - T_j^{\text{off}} > 0 \Rightarrow \Phi_j \leq \Phi_{k+1}$$

Since user U_{k+1} can offload, i.e., $\Phi_{k+1} \leq 0$, we have $\Phi_j \leq \Phi_{k+1} \leq 0$. ∎

Theorem 2. *If user U_{k+1} cannot offload in iteration $k + 1$, then user U_j cannot offload either, for all $j > k + 1$.*

Proof: Recall that during iteration $k+1$, and for any user $j > k + 1$, we have that $a_j = 0$ and

$$T_j^{\text{off}} = \sum_{i=1}^{j-1} a_i\beta_i + (1 + \sum_{i=j+1}^{n} a_i)\beta_j = \sum_{i=1}^{k} \beta_i + \beta_j.$$

Therefore (11) implies that

$$T_{k+1}^{\text{off}} - T_j^{\text{off}} \leq 0 \Rightarrow \Phi_j \geq \Phi_{k+1}$$

Since user U_{k+1} could not satisfy its time constraint, we have $\Phi_{k+1} > 0$, and, therefore, $\Phi_j \geq \Phi_{k+1} > 0$. ∎

Theorems 1, 2 imply that any decision change for a user incurred by the algorithm does not result in simultaneous changes for other users. Hence, when the algorithm terminates there is no user that would like to change its decision unilaterally, i.e., we have reached a Nash equilibrium.

Definition 2. *An MCC system is* homogeneous *iff*

$$\beta_i = \beta_j \textit{ and } \tau_i = \tau_j, \quad \forall i, j \in \{1, \ldots, n\}.$$

Since an homogeneous system is also semi-homogeneous, a Nash equilibrium can always be found in the same way as above.

VI. Experimental Results

To evaluate the efficiency of the game theoretic model, the convergence time and the energy consumption attained at the Nash equilibrium points (NEP) is compared to the social optimum cost. Our simulation results demonstrate that the Gauss-Seidel method always converges to a Nash equilibrium point, in both homogeneous and heterogeneous parameter cases. In order to cover a wide range of scenarios, multiple random configurations were generated and each was executed multiple 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, were chosen randomly between 1 and 10 Gcycles. Input data size, B, is between 0.42 to 42 Mb and the channel data rate, r, ranged from 6.4 to 64 Mbps. Local computation power, f^l, 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, was between 0.75 to 1 watts. Local energy consumption, v^l, is considered to be equal to P^l/f^l and local execution power consumption, P^l, was chosen randomly from 20 , 22.5 and 25 watts.

Figure 2 shows the number of discovered Nash equilibrium points. Generally, we can see that the number of points increases with the number of users which is consistent with what would be expected. In this experiment, five hundred random configurations were generated and each configuration was executed one hundred times with a random initial decision vector and random user selections with different seeds. For small numbers of users (up to 12) we can see a linear increase in the number of NEPs. To cover as many scenarios as possible, we repeated this experiment more than 50000 times and that is the reason that we can see large changes for $N > 12$.

In Figure 3 the average and maximum game convergence time is shown, which increases linearly with the number of users. Computing the centralized optimal computation offloading solution involves solving the integer linear program which is very time consuming. This shows that the game theoretic computation offloading mechanism scales well with the size of the problem. For this experiment, five hundred configurations were generated and each random configuration was executed one hundred times with random initial decision vectors and user selections.

Figure 4 illustrates the average offloading ratio value in 10000 runs, which is defined as the ratio of the number of remote executions to the number of users, as well as the variation between the maximum and minimum ratio for each system population. As the number of users increases, proportionally fewer users end up offloading at equilibrium. This is expected since the channel capacity is kept constant, and,

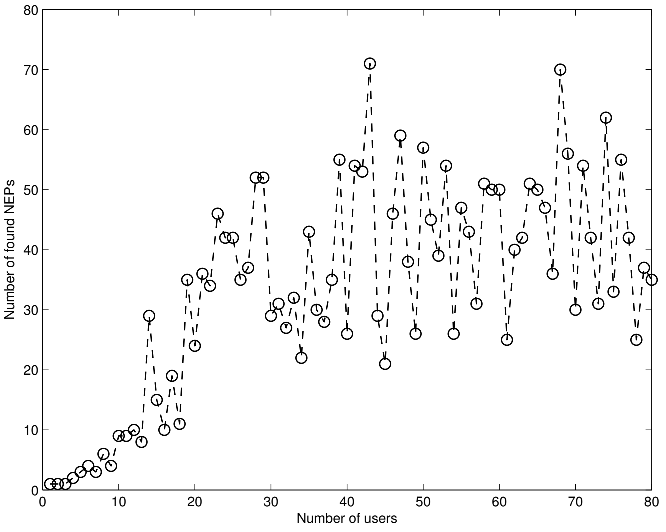

Fig. 2. Number of Nash Equilibria

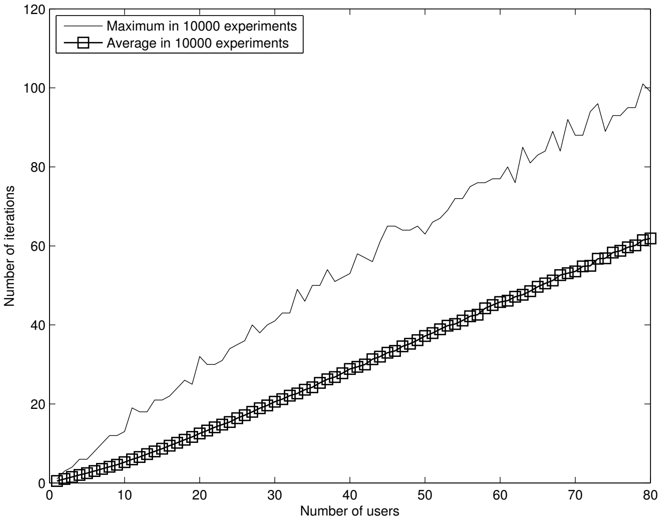

Fig. 3. Number of Iterations vs. Number of Users

therefore, the remote execution delay becomes prohibitively bigger for an ever greater proportion of users.

To calculate the ratio of overall energy consumption to the social optimum, we generated 500 random configurations for each number of users. In order to estimate the worst Nash equilibrium we ran the Gauss-Seidel algorithm 100 times for each configuration with random initial decision vectors and different random seeds. Figure 5 shows the ratio of the total cost at equilibrium over the optimal social cost (OPT). More specifically, we show the ratio for the worst (total cost-wise) equilibrium reached, the best (total cost-wise) equilibrium reached, and the average over all reached equilibria. This was done by generating 500 random configurations for each number of users and then running the Gauss-Seidel algorithm 100 times for each configuration, with random initial decision vectors and different random seeds. We note that in our case

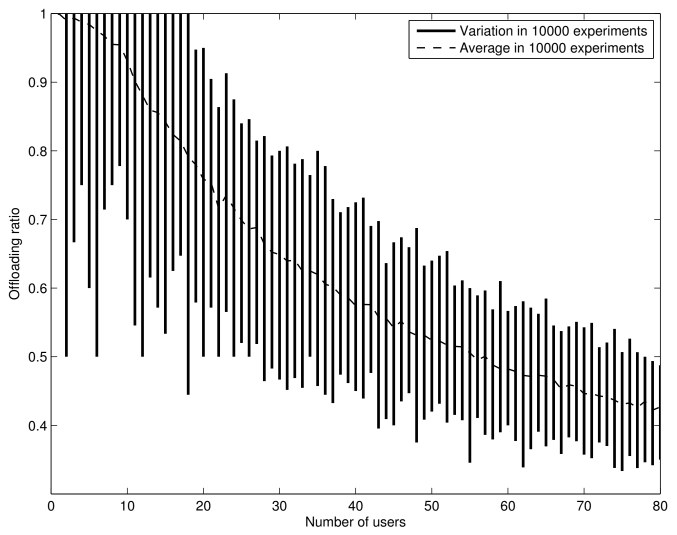

Fig. 4. Offloading Ratio versus Number of Users

the best equilibrium cost is virtually the same as the social optimum (the ratio is 1), but we have examples where this is not true. While the cost for the worst equilibrium may be even 16% 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 doesn't result in a prohibitive increase of the total energy needed for supporting offloading. We leave open the question of theoretical upper bounds for the worst-case equilibrium ratio (i.e., the PoA as defined by [15]).

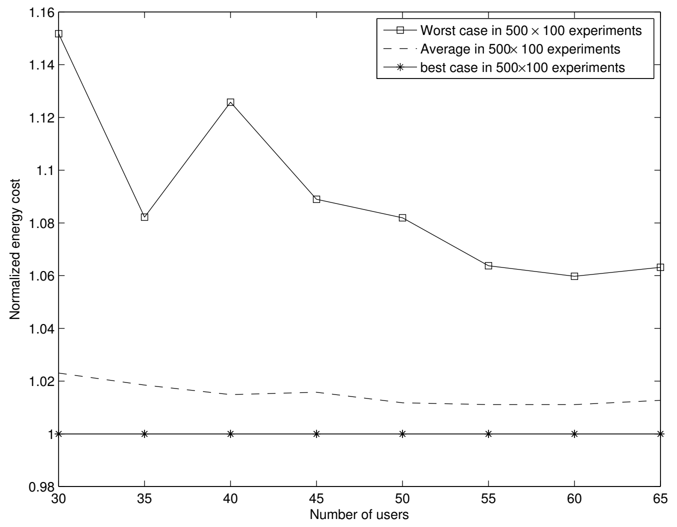

Fig. 5. Normalized Social Energy Consumption

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 which 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 which illustrate that the system always converges to a Nash equilibrium using the Gauss-Seidel method. Data was also presented which shows the number of Nash equilibria that were found, the number of iterations required, and 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*. 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] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] X. Chen, "Decentralized Computation Offloading Game for Mobile Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[14] 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.

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