A Game Theoretic Resource Allocation for Overall Energy Minimization in Mobile Cloud Computing System[1]

Yang Ge, Yukan Zhang and Qinru Qiu
Department of Electrical Engineering and Computer Science, Syracuse University
Syracuse, NY, USA, 13210
{yage, yzhan158, qiqiu}@syr.edu

Yung-Hsiang Lu
Department of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA, 47907
yunglu@purdue.edu

ABSTRACT

Cloud computing and virtualization techniques provide mobile devices with battery energy saving opportunities by allowing them to offload computation and execute code remotely. When the cloud infrastructure consists of heterogeneous servers, the mapping between mobile devices and servers plays an important role in determining the energy dissipation on both sides. From an environmental impact perspective, any energy dissipation related to computation should be counted. To achieve energy sustainability, it is important reducing the overall energy consumption of the mobile systems and the cloud infrastructure. Furthermore, reducing cloud energy consumption can potentially reduce the cost of mobile cloud users because the pricing model of cloud services is pay-by-usage. In this paper, we propose a game-theoretic approach to optimize the overall energy in a mobile cloud computing system. We formulate the energy minimization problem as a congestion game, where each mobile device is a player and his strategy is to select one of the servers to offload the computation while minimizing the overall energy consumption. We prove that the Nash equilibrium always exists in this game and propose an efficient algorithm that could achieve the Nash equilibrium in polynomial time. Experimental results show that our approach is able to reduce the total energy of mobile devices and servers compared to a random approach and an approach which only tries to reduce mobile devices alone.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, andserviceability

General Terms

Algorithms, Management, Performance

Keywords

Congestion Game, game theory, mobile cloud computing, power management, virtualization

1. INTRODUCTION

The emerging paradigm of mobile cloud computing (MCC) moves the processing, memory and storage requirements all together from the resource limited mobile devices to the resource unlimited cloud. MCC provides many advantages to the mobile devices [3]. It extends the storage capacity for mobile users [4] and also reduces the risk of data and application lost on mobile device by backing up users data on several computers in the cloud. Security services such as virus scanning and malicious code detection provided by the MCC improves the safety and reliability of the mobile device.

One very important benefit brought by MCC for mobile users is the extended battery life time. The MCC helps the mobile devices to run the computation intensive applications, which normally consume a large amount of battery energy. This is enabled by virtualization technique which allows the cloud infrastructure to run arbitrary mobile applications from the mobile users or service subscribers. We refer to this technique as *computation offloading*. Reference [5] presents a high level analysis on the conditions that computation offloading could save the energy for mobile phones. According to their results, an application with large amount of computation but very limited data communication could benefit most from computation offloading. The authors in [6] propose an architecture called MAUI to dynamically control the computation offloading for .NET applications at runtime. MAUI utilizes some .NET features to partition and profile the applications and formulate the offloading problem as a linear programming (LP) problem. The authors in [7] propose a similar architecture for Android applications.

Although moving the computation energy away from the mobile devices and into the cloud relieves the pressure on the devices' batteries, it will increase the energy consumption of cloud infrastructure such as the servers in the data center. From environmental impact and carbon emission control perspective, energy is task-centric instead of system-centric [1]. Any energy dissipation related to computation should be counted and carefully managed. From cost reduction point of view, reducing energy consumption of cloud can potentially reduce the cost of mobile users, because current pricing model of cloud services is pay-by-usage and energy consumption is a major factor in the operating cost of cloud services.

In this paper, we consider the problem of energy minimization for a mobile cloud computing system under computation offloading. The MCC system consists of a group of mobile devices and a set of servers in the data center. Each mobile device runs an application and tries to upload a portion of its application to one of the servers. The offloading strategy involves two decisions, (1) the amount of computation to be offloaded, and (2) the destination of offloading. A mobile device selects offloading strategy not only to minimize its own energy dissipation, but also to minimize its energy usage in the cloud. The decision of one mobile device changes the status of the cloud infrastructure, for example one particular server will become more congested or wakeup from sleep mode. Since each mobile device chooses its offloading strategy to minimize the overall energy, a status change in the cloud infrastructure will trigger some mobile devices to adjust their offloading strategies. This, consequently, will impose more

[1]This material is based upon work supported by the National Science Foundation under Grant No. CNS-0845947

changes in the cloud and leads other mobile devices to adjust their offloading strategies. From the above analysis an interesting question is raised, if all mobile devices aim at minimizing the end-to-end overall energy and adjust their offloading strategies independently, will the system ever become stable?

We formulate the MCC system energy minimization problem as a class of games called congestion games. In this model, each mobile device is a player and his strategy is to select one of the available servers to offload its computation. All players compete for the same set of resources and their goal is to minimize the combined mobile and server energy dissipated to provide service for its own application. We prove that the Nash equilibrium always exists in this congestion game formulation. Nash equilibrium is the optimum policy in the sense that no player can find better policy if he deviates from current policy unilaterally [2]. We propose an efficient algorithm that achieves the Nash equilibrium in polynomial time.

There have been some works in the literature using game theory to solve general resource allocation problem in cloud computing environment, for example [14]. However, this work only considers the cost in cloud side and ignores the cost on mobile devices. Furthermore, it defines cost as a simple linear function of workload, which is not a suitable model of energy dissipation if we want to capture some nonlinearity and discontinuity introduced by server power management.

The uniqueness of our work is summarized as the following:

- This is the first work that aims to reduce the overall energy of a mobile cloud computing system under computation off-loading context. Although computation offloading techniques have been investigated in some previous work [5][6][7], their goals are only to reduce the energy consumed by the mobile devices and extend their battery lives, without considering the energy consequence of offloading on the cloud computing infrastructure.
- We proposed a game theoretic formulation of the problem. We also proved that the Nash equilibrium of this problem always exists and it could be achieved in polynomial time if each mobile device selects its offloading strategy based on our proposed algorithm. The important implication of Nash equilibrium is that our algorithm will eventually converge to a stable state, where every mobile device finds its current optimal strategy and has no incentive to leave.
- We demonstrate the necessity of joint optimizing the energy of the mobile devices and the cloud infrastructure. Compared to the techniques that only aim to reduce the mobile device energy, our technique is able to reduce the overall energy by 45.42%. Our approach is also able to reduce more than 61.83% energy compared to a random offloading approach.

The rest of the paper is organized as follows: We discuss our system model and introduce the energy minimization problem in Section 2. We introduce the congestion game model and discuss its application in detail in Section 3. We present the algorithm to achieve the equilibrium in Section 4. Experimental results are reported in Section 5. Finally, we conclude the paper in Section 6.

2. SYSTEM MODEL

2.1 MCC System Architecture

On one side of the MCC architecture, mobile devices like smart phones, tablet computers are connected to the cloud through Wi-Fi or 3G networks. On the other side, a group of servers residing in a data center constitute a large distributed computing system which can provide the cloud users with different kinds of services, including infrastructure as a service (IaaS), platform as a service (PaaS) or software as a service (SaaS) [3]. In this paper, we focus on the IaaS, in which the servers in the cloud provide the mobile devices with hardware resources, like CPUs and memories for them to offload computation for battery energy saving.

At a specific execution point of the mobile application, the migration manager decides to move a portion of the application to the cloud (for example, an individual thread). The manager will send the migration request as well as necessary data and program states to the remote server. Upon receiving the migration request, the application data and program states, the server creates a dedicated virtual machine (VM) for the mobile device, loads the application executable and starts execution. In the mean time, the mobile device continues to run other threads or waits for the results return from the remote server. At the end, the migrated portion returns back to the mobile device, and merges back to the original process.

Mobile devices generally can be benefited from code offloading and remote execution because of two reasons. Firstly, the plenty of hardware resources in the cloud can help mobile devices to overcome resource limitation and run some resource intensive applications. Secondly, by executing code remotely, the mobile devices could avoid spending a long time in high power state, and either stay in idle state or go into low power sleep state, thus save the energy and extend the battery life. However, as pointed out in [5], not all kinds of mobile applications could save energy through code offloading. For example, an application with very high data communication volume will spend much more extra energy in transferring the data between local memory and remote servers and not benefit from computation offloading for energy saving. Deciding which portion of the application needs to be offloaded is non-trivial. It usually needs detailed program profiling information and a fast solver for integer linear programming problems. In this paper, we assume that each device has made their own decisions about which part of the mobile applications should be offloaded to the cloud. We focus on how a mobile device selects a server from a group of heterogeneous servers in the data center for computation offloading.

2.2 MCC Energy Minimization Problem

We assume that there are n mobile devices and r cloud servers in the MCC system. The ith mobile device runs an application A_i with computation workload C_i (which can be measured by the number of clock cycles or execution time). We assume that, based on careful application profiling and device characterization, the amount of computation to be offloaded to the dedicated virtual machine VM_i is pre-determined and is denoted as O_i. The amount of computation left for local execution is denoted as L_i, $L_i + O_i = C_i$. We also assume that the mobile device has a performance requirement on VM_i, i.e. the O_i amount computation must be finished within deadline D_i, otherwise offloading computation to server will not bring notable performance benefit. If this performance constraint cannot be satisfied, the mobile device will either reduce the amount of offloading, or move the VM to a faster server or even request to wake up a new server.

We assume the power consumption of the mobile device i during the execution of the application A_i is determined by an application specific factor α_i, which is a high level parameter reflecting its overall power intensity, i.e. $P_{active,i} = \alpha_i P$, where P is the normalized maximum mobile device power consumption.

Therefore, the energy consumed by running the local portion of the application A_i is $E_{local,i} = P_{active,i}L_i$.

We assume that the servers in the data center are also heterogeneous machines. They have different processing speeds and power consumption, so they could accommodate different service requests (delay sensitive or power sensitive) from the mobile clients. Here we assume server j is operating at speed s_j and its power consumption is $f(s_j)$, e.g. $f(s_j) = s_j{}^3$. Please note that our algorithm does not rely on any properties of f. We assume different offloaded applications have different power even running on the same server. Similar to the factor α_i on the mobile device, we assign each application a parameter β_i. So the power consumption of application A_i running on server j will be $\beta_i f(s_j)$. And the energy consumption for the jth server to run the offloaded code from device i will be calculated as $E^i_{server,j} = \beta_i f(s_j)O_i$.

If only one application A_i running on server j, we assume that the waiting time experienced by the mobile device i is $g(s_j)O_i$, where $g(s_j)$ is a non-increasing function of speed s_j. If there are n_j applications running on the server j, we assume the waiting time experienced by the mobile device i is $g(s_j)O_i h_j(n_j)$, where $h_j(n_j)$ is a non-decreasing function of n_j. For example, if each VM has equal time slice on the server, without considering the overhead of context switch and cache miss, $h_j(n_j)$ can be simplified to n_j. As a server gets more congested, A_i running on it will experience longer delay and might violate the performance constraint D_i. The mobile device has to reduce the computation amount O_i to satisfy D_i or selects another server which is less congested. [12] shows that, with proper configuration, network latency in a data center is independent of network topology and server's location. Thus we ignore the network latency in this problem formulation. Neither do we consider the communication energy on the routers and switches because they are usually small comparing to the energy consumed on mobile devices and servers [13], and they are not affected by the mapping of the computation.

The overall energy related to the execution of application A_i consists of the energy dissipation on both the server and mobile device. It can be calculated as the following:

$$E_{total}(i,j) = E_{local,i} + E^i_{server,j} \qquad (1)$$

j is the index of the server to which the application A_i is mapped.

We also assume servers follow time out power management policies, i.e. they will switch to low power mode after a certain period of idle. Let Tdenote the time out threshold and $P_{static,j}$ denote the static power server j consumes when it is idle. Then the static energy consumption of server j during the idle period is $E_{static,j} = P_{static,j} \times T$. The total energy consumed by the MCC system is $E_{MCC} = \sum_{i=1}^{n} E_{total}(i) + \sum_{j=1}^{r} E_{static,j}$.

The MCC system energy minimization problem can be stated as the following resource allocation problem:

Given n mobile devices and r servers, for each device, find a server for computation offloading, such that the overall MCC system energy E_{MCC} is minimized while the performance constraint is satisfied.

This problem can be formulated as an integer linear programming (ILP). When the number of mobile devices and cloud servers gets large, which is usually the case in reality, solving such ILP becomes exponentially difficult. In the next sections, we introduce a game theoretic formulation which solves this problem using a distributive approach, where each mobile device chooses its own offloading strategy, including the amount of offloading and the destination of offloading in order to minimize its overall energy.

3. GAME THEORETIC FORMULATION

3.1 Congestion Game Model

Congestion games [8] model a group of players sharing a set of resources. In a congestion game, each player chooses one or a subset of resources to maximize his own utility or minimize his own cost. The utility/cost obtained by the player is the sum of the utilities he received from each resource he chooses. The utility/cost received from a resource depends on how many players sharing the same resource and it is generally a non-increasing/non-decreasing function of the number of players sharing it. For example, the more senders in a network share the same link to send packets, the less throughput and longer waiting time they will experience. In this paper, we consider a simple congestion game in which each player only selects one resource, because each mobile device is only allowed to offload its computation to one server.

We introduce the following notation to define congestion game.

- $N = \{1, \dots, n\}$ is a finite set of n players
- $R = \{1, \dots, r\}$ is a finite set of r resources
- Σ_i is the set of strategies for player i, which is a subset of resource set R. We use $\Sigma = \prod_i^n \Sigma_i$ to denote the joint strategy space and $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$ a strategy tuple in which the player i plays strategy σ_i, i.e. it chooses resource $\sigma_i \in \Sigma_i$. Please note σ is the strategy vector, and σ_i with subscript i is the strategy of player i. Sometime we also use J to denote a strategy in our algorithms.
- n_j is the number of players who selects strategy j, and for a strategy tuple σ, $n_j = \sum_{i=1}^{n} I(\sigma_i = j)$, I is the indicator function. We call $(n_1, n_2, \dots, n_r)$ a congestion vector.
- S_{ij} is the cost function when player i selects resource j. S_{ij} is a monotonically non-decreasing function of n_j, i.e. $S_{ij}(n_j) \le S_{ij}(n_j + 1)$ for any positive integer n_j.

For some specific congestion games, there exists Nash equilibrium. Nash equilibrium is a general term in game theory and is defined as a state that no player can benefit by changing its strategy while the other players keep their strategies unchanged [1]. For a congestion game defined as above, a strategy tuple σ leads to Nash equilibrium if and only if

$$S_{i\sigma_i}(n_{\sigma_i}) \le S_{ij}(n_j + 1), \forall 1 \le i \le n, 1 \le j \le r$$

σ_i is also said to be the *best reply* or *best response* for player i against strategy tuple σ. In other words, a strategy vector σ is at Nash equilibrium, if for any user i, moving away from resource σ_i to choose resource j by itself will lead to higher cost.

3.2 Application to MCC Energy Minimization Problem

To apply the congestion game model to our MCC system energy minimization problem, we define each mobile device and the dedicated VM created for it to be a player and each server to be a resource. We use mobile device, VM, player interchangeably in the paper. For each player, the strategy set Σ_i is equal to the resource set R, and if mobile device i selects server j, then $\sigma_i = j$.

One way for the player (i.e. the mobile device) to define its cost function is to use the total energy consumed locally. By reducing his own energy, the player can extend its battery life. However, if every player wants to save his own local energy, he will select faster servers to maximize the offloaded computation. This makes the faster servers more congested than the others and as a result, there might be performance violation and the mobile device have to either reduce the offload amount or move to another server. Furthermore, our objective is not to minimize the mobile devices' energy but the overall system energy. We would like each player be less selfish and take some "social responsibilities". Therefore, we define the first component of cost function as the sum of weighted energy associated with the execution of application A_i at both mobile device side and server side, as show in equation (2).

$$a_i E_{local,i} + b_i E^i_{server,j} \quad (2)$$

By this definition, each player does not only try to minimize its own energy but also the energy that he added on the servers as well. The weight coefficients a_i and b_i in this equation differentiate the *quality* of the energy used by a mobile device and a server. The former, due to the overhead of battery charge and discharge and also because of the limited battery capacity, is usually considered higher quality, therefore we have $a_i \geq b_i$. These weight coefficients do not have to be the same for all mobile devices. For example, when a mobile device's battery is full, it could be more altruistic and reduce the value of a_i to offload less computation; when its battery drops under a critical level, it could increase the weight on $E_{local,i}$ and offload more computation.

Eq. (2) is a non-decreasing function of n_j. Whenn_j increases, server j becomes more congested. As a result, less offloaded computation can be completed within the performance constraint and more may have to be done on the mobile device locally, thus gives an increase in $E_{local,i}$ and decrease in $E^i_{server,j}$. Because servers usually have better energy efficiency than the mobile devices due to their multi-core architecture and powerful co-processors such as GPUs and DSPs, the increase in $E_{local,i}$ is usually greater than the decrease in $E_{local,i}$. Also because coefficient a_i is greater than or equal to coefficient b_i, Eq. (2) is a non-decreasing function of n_j.

In order to consider each server's static energy penalty during the time-out period, we add the second component to each player's cost function. It is proportional to the current total static energy, i.e. $w_i(\sum_{j=1}^{n} E_{static,j})$, w_i is a constant weight. Therefore the overall cost function for a player is as following:

$$S_{ij} = a_i E_{local,i} + b_i E^i_{server,j} + w_i(\textstyle\sum_{j=1}^{n} E_{static,j}) \quad (3)$$

Because the static penalty does not depend on the congestion of each server, the overall utility is still a non-decreasing function of n_j. The cost function meets the requirement of a congestion game.

4. NASH EQUILIBRIUM OF THE ENERGY MINIMIZATION CONGESTION GAME

In this section, we present the algorithms which find the Nash equilibrium for the energy minimization problem defined above. The algorithms themselves provide a constructive proof for the existence of the Nash equilibrium. Please note that the actual algorithm to achieve the equilibrium could run by the dedicated VM to relieve the computation burden on mobile devices and reduce communication between mobile devices and the cloud.

Our algorithm adopts an incremental optimization scheme. At each step, only one player is allowed to change its current strategy and choose its best response against other players' current strategies. Consider the initial scenario that the first $n-1$ players have achieved equilibrium and their strategy vector is $\sigma(0) = (\sigma_1(0), \sigma_2(0), \dots, \sigma_{n-1}(0))$. We are interested to find out, after the nth player enters, how will the system regain the equilibrium.

Let R denote the set of all servers and S_{on} denote the set of servers which have already been turned on. Let $J(0) = \sigma_n(0)$ denote the first strategy chosen by player n against the initial strategy vector $\sigma(0)$ of the rest of the system. We consider the following two scenarios.

Scenario 1: $J(0) = \sigma_n(0) \in S_{on}$. Then the only players affected by player n's choice are those players whose current strategy is also $J(0)$ in $\sigma(0)$, because player n makes server $\sigma_n(0)$ more congested. For other players, their current strategies remain to be their best strategies. If every player in server $J(0)$ finds $J(0)$ remaining to be its best choice, then Nash equilibrium is achieved. Otherwise, there will be a player p on $J(0)$ whose best strategy deviates from its current strategy. Assume this player moves from $J(0)$ to $J(1)$ at step 1. We let the process continue as shown in algorithm1. The algorithm returns when all players are in their best strategies or a new server is turned on. Please note that in the algorithm, $J(i)$ represents a strategy that a player chooses at step i. It can be proved that this process will stop in finite step. The claim is given in Lemma1. We skip the proof due to space limit.

Algorithm1: type1_ move **($\sigma_n(0), \sigma(0), R$)**

```
1.  $i = 0; J(0) = \sigma_n(0)$;
2.  while true
3.      $p = null$;
4.      for each player $p$ with strategy $\sigma_p(i) = J(i)$
5.          if $J(i)$ is not player $p$'s best reply against $\sigma(i)$ in $R$
6.              $J(i+1)$ = player $p$'s best reply against $\sigma(i)$ in $R$
7.              $\sigma_q(i+1) = \sigma_q(i), \forall q \neq p$,
8.              $\sigma_p(i+1) = J(i+1)$;
9.              break;
10.         end
11.     end
12.     if ($\nexists p$) return $\emptyset$;
13.     else if ($J(i+1) \notin S_{on}$)
14.         turn on $J(i+1)$;
15.         return $J(i+1)$;
16.     end
17.     $i$++;
18. end
```

Lemma1: TYPE1_MOVE will stop in at most n steps, where n is the number of players. Furthermore, if type1_move returns at line 12, then Nash equilibrium is achieved.

We define a limited version of type1_move called type1_move_limited. The limited version restricts the move in a given subset of servers denoted as S_{limit}. We set the third input of type1_move algorithm to S_{limit} instead of using R in the limited version.

Algorithm2: type2_ move **($J(1), \sigma(0), R$)**

```
1.  $i = 1$;
2.  while true
3.      $p = null$;
4.      for each player $p$ with strategy $\sigma_p(i) \neq J(i)$
5.          if $J(i)$ is player $p$'s best reply against $\sigma(i)$ in $R$
6.              $\sigma_q(i+1) = \sigma_q(i), \forall q \neq p$,
7.              $\sigma_p(i+1) = J(i); J(i+1) = \sigma_p(i)$;
8.              break;
9.          end
10.     end
11.     if ($\nexists p$)
12.         break;
13.     end
14.     $i$++;
15. end
```

Scenario 2: If $J(0) = \sigma_n(0) \notin S_{on}$. Then all players' strategies will be affected by the newly turned on server because of two reasons. Firstly, all players' utilities are increased by the static energy penalty of turning on $J(0)$. Secondly, $J(0)$ is much less congested than other servers in S_{on}, players could potentially offload more computation if they choose $J(0)$ and achieve energy savings (reduce cost function). For each player, if the current strategy is its best strategy then the system is still in equilibrium; otherwise $J(0)$ will be its best strategy. Assume player p_1 on server $J(1) = \sigma_{p1}(0)$ finds its current best response to be $J(0)$. Then at step 1, we let player p_1 move from $J(1)$ to $J(0)$, i.e. $\sigma_{p1}(1) = \sigma_n(0)$. p_1's move will make server $J(1)$ less congested than before. Then there might be a player p_2 on $J(2) = \sigma_{p2}(1) \neq J(1)$ find its best strategy to be $J(1)$. Then at step 2, we let p_2 moves from $J(2)$ to $J(1)$. We let the process continues and summarize it in algorithm2. We call it type-2 move. We claim type-2 move will stop after finite steps in Lemma2. Again the proof is skipped due to space limit.

Lemma2: TYPE2_MOVE will stop in at most $n \cdot r_{on}$ steps, where r_{on} is the number of servers which has been turned on.

Again, we define a limited version of type2_move called type2_move_limited. The limited version restricts the move in a given subset of servers denoted as S_{limit}. We set the third input of type2_move algorithm to S_{limit} instead of using R in the limited version.

We now present our algorithm for finding Nash equilibrium for the first n player. We again assume the first $n-1$ players have achieved equilibrium. Lemma1 shows that if $J = \emptyset$ (line 8), then the equilibrium is achieved. If $J \neq \emptyset$, from algorithm1 we know that the congestion vector for servers in S_{on} does not change after type1_move (line 5). Because the cost function (3) is an increasing function of the congestion, for each player, either the current strategy is its best strategy or newly turned on server J is its best strategy. Property 2 in Lemma3 shows that the loop 11 ~ 15 will repeat at most n iterations. After this loop, property 1 show that the first $n-1$ players are in their best strategies and player n is either in its best strategy or has its best strategy outside S_{on}. Because there are only r servers in the system, the outer loop will repeat at most r times. After the outer loop terminates, all players are in their best strategy and Nash equilibrium is achieved. We skip the proof of Lemma3 due to space limit.

Algorithm3: find_equilibrium_for_first_n_players(n)

```
1.  while true
2.      $J$ = best strategy of player $n$; $\sigma$ = current strategy vector;
3.      assign player $n$ to $J$;
4.      if ($J \in S_{on}$)
5.          $J$ = type1_move($J, \sigma, R$);
6.      end
7.      if ($J == \emptyset$)
8.          return;  // reach the equilibrium
9.      end
10.     $S_{on} = S_{on} \cup J$;  // turned on a new server $J$
11.     while ($\exists p$, its current strategy is $\sigma_p$ and its best strategy is $J$)
12.         move $p$ from $\sigma_p$ to $J$;
13.         type1_move_limited($J, \sigma, S_{on}$);
14.         type2_move_limited($\sigma_p, \sigma, S_{on} - J$);
15.     end
16.     if (player $n$ is in its best strategy)
17.         return;  // reach the equilibrium;
18.     end
19. end
```

Lemma3: The loop in line 11 ~ 15 have the following properties:

1. After every iteration, for each of first $n-1$ player p, either it is in its best strategy or new server J is its best strategy.
2. The number of players in the server J keeps non-decreasing. If a player p leaves J at an iteration, it never comes back to J.
3. After the loop, for player n, either the current strategy is its best strategy or its best strategy is not in S_{on}.

5. EXPERIMENTAL RESULTS

To demonstrate the effectiveness of our approach, we implement our Nash equilibrium algorithm in Matlab and carry out the experiments on a DELL T3400 workstation. We obtain the power model of mobile applications from [9], which is a linear combination of the activities on different components, including CPU utilization, Wi-Fi module status, the brightness of the LCD etc. Because our purpose is not a detailed power analysis for mobile devices, we just use one factor α to represent the overall activities of the application. We assume the maximum power consumption of a mobile device is 2.4W and the actual power of the application is $P_{app} = \alpha \times 2.4\text{W}$. We set the activity factor α as a random variable with a uniform distribution in the range $[0.3, 1.0]$. We assume the offloaded computation amount is predetermined by careful application profiling as in [6][7].

We assume the servers are heterogeneous and they are operating at different speeds in the range of [2.0GHz, 3.0GHz]. We adopt the server power model from [15], which is based on Intel Xeon processors. According to this model, server power is a linear function against the CPU's frequency at full utilization. It is 200W at 2.0GHz and 240W at 3.0GHz. These data are the full system power including the static power. Based on the data provided in [16], we assume the static power of our server is 100W when it is in idle state. Reference [10] shows that at a given operating frequency, server power is a linear function against the VM system utilization β. We set $\beta = \alpha$ in our experiments for simplicity. Overall, we assume the server power is a linear function against both operating frequency and utilization. This is also confirmed by [17]. At 100% system utilization, the server consumes 100W to 140W more power than in idle state.

For the delay model, we assume all servers provide M times speedup over the mobile devices [5] when running at 2.0GHz. When running at speed s, its speedup ratio is $sM/2$, i.e. $g(s_j) = 2/(s_j M)$. If there are N mobile devices offloading to the server, we assume the slowdown factor is $h(N) = N^\gamma, \gamma \geq 1$, γ accounts for the context switching overhead [11]. If a mobile device i offloads an amount of 10 seconds computation on a server with speed s and shared by N mobile devices, device i will wait for $10 \times (sM/2) \times N^\gamma$ time to get results. In our experiment, γ is set to 1.05. Please note that our optimization algorithm does not rely on any specific power or delay models.

5.1 Results Analysis

In the first set of experiments, we compare our policy with a policy which randomly assigns the mobile devices to cloud servers. We call this policy *Random*. The proposed policy that minimizes the overall energy is denoted as *Nash-overall*. We set 10 servers and 60 mobile devices in the MCC system. The results are from the average of 10 runs.

Table 1 Comparison between Nash-overall and Random

Mobile weight (a)	Nash overall (J)	Random (J)	Improvement (%)
1	3558.26	9319.07	61.83%
1.5	5695.91	9319.07	38.92%
2	6007.31	9319.07	35.55%
2.5	6637.01	9319.07	28.79%

Table 1 shows the energy comparison between Nash-overall policy and Random policy. We vary the mobile weight, i.e. a in equation (3) from 1 to 2.5. As the a goes bigger, the Nash-overall algorithm tries to complete more offloaded computation on the server side. The random policy is not affected by the weight. As shown in this table, our Nash-overall policy could achieve large energy savings compare to the Random policy. This is because our policy tries to reduce the static energy consumption of the servers by conservatively waking them up and consolidating the offloaded computation to a small number of energy efficient servers, especially when the weight   is small. On the other hand, when   is big, the first component in equation (3) becomes dominant. Our policy tries to reduce the energy on mobile devices and turn on more servers to accommodate the offloaded computation. Therefore, the static energy increases which causes higher total energy consumption.

This trend can be seen in Figure 1, which shows the energy components of each policy. The blue bar represents the dynamic energy when servers are running the offloaded computation and the red bars represent the static energy when servers. Because the Random policy will blindly turn on servers, it incurs most static energy. However, this behavior is beneficial for mobile devices because mobile devices can be uniformly distributed among all the servers, and each server is less congested. Servers could complete more offloaded computation under mobile devices' performance constraints. Therefore, the Random policy results the minimum mobile devices energy consumption among all five groups. For our policy, when weight   is small, mobile device energy is dominant, when   is big, the server static energy is dominant. As shown, when   , mobile devices' energy is same as the Random policy. Further increasing   does not reduce the mobile devices' energy anymore because all offloaded computation has been completed by the servers. In this case, we still achieve 28.79% total energy savings compare to Random policy, because each mobile device is aware of server static energy and will not turn on unnecessary servers. It is the unique feature of our algorithm to get energy tradeoff between mobile devices and cloud servers.

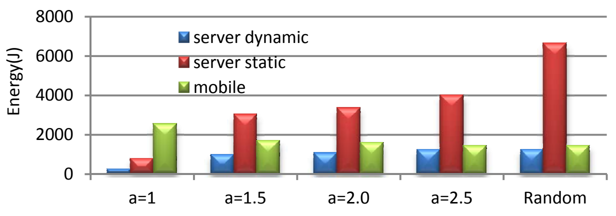

Figure 1 Energy components of Nash-overall and Random policy

Table 2 Comparison between Nash-overall and Greedy

Mobile weight (a)	Nash overall (J)	Greedy (J)	Improvement (%)
1	3659.28	6703.22	45.42%
1.5	5779.22	6703.22	13.80%
2	6153.47	6703.22	8.18%
2.5	6575.44	6703.22	1.90%

In the second set of experiments, we compare Nash-overall policy with a greedy policy which tries to reduce mobile devices energy. For a mobile device, this greedy policy first selects those servers which could mostly satisfy its offload computation within the performance constraint. Among these servers, the greedy policy then selects the server which could minimize its total energy. Table 2 shows the energy comparison between Nash-overall and Greedy policy as the weight   gradually increases. This table shows similar trend as in Table 1, because the greedy policy is also relatively aggressive in turning on servers. When   is small, the total energy reduction is most significant. When   increase to 2.5, the reduction in total energy decrease from 45.42% to 1.9%, because the Nash-overall also become more aggressive in turning on new server and incurs larger server static energy. On the other hand, the energy consumption differences on mobile devices between these two policies decrease from 43.75% to only 1.64%.

6. CONCLUSIONS

In this paper, we propose a game-theoretic approach to optimize the energy consumption of the MCC systems. We formulate the mobile devices and cloud servers energy minimization problem as a congestion game. We prove that the Nash equilibrium always exists in this congestion game, and propose an efficient algorithm that could achieve the Nash equilibrium in polynomial time. Experimental results show that our approach is able to reduce the total energy compare to a random approach and an approach which only tries to reduce mobile devices energy.

7. REFERENCES

[1] Y. H. Lu, Qinru Qiu, A. R. Butt and K. W. Cameron, "End-to-End Energy Management," in *IEEE Computer*, 44 (11), November 2011.

[2] M. Osborne and A. Rubinstein, "A Course in Game Theory" Cambrisdge, MA: MIT, 1994.

[3] H. Dinh, C. Lee, D. Niyato and Ping Wang, "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches", in *Wireless Commu. and Mobile Comput.*, 11(11), Oct. 2011.

[4] http://aws.amazon.com/s3/

[5] K. Kumar and Y. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" in *IEEE Computer*, vol. 43(4), Apr. 2010.

[6] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *Proc. of international conference on Mobile systems, applications, and services,* 2010.

[7] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: elastic execution between mobile device and cloud," in *Proc. of conf. on Computer systems (EuroSys)*, Apr. 2011.

[8] I. Milchtaich, "Congestion games with player-specificpayoff functions," in *Games and Economic Behavior*,13(1):111–124, 1996.

[9] L. Zhang , B. Tiwana, Z. Qian , Z. Wang , R. Dick , Z. Mao and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. of Inter. Conf. on HW/SW codesign and system synthesis*, Oct. 2010.

[10] M. Pedram and I. Hwang, "Power and Performance Modeling in a Virtualized Server System," in *Proc. of International Conference on Parallel Processing Workshops* (*ICPPW*), Sep. 2010.

[11] T. Enokido, A. Aikebaier and M. Takizawa, "A Model for Reducing Power Consumption in Peer-to-Peer Systems," in *IEEE System Journal*, vol. 4, issue 2, pp. 221-229, Jun. 2010.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel and S. Sengupta. "VL2: a scalable and flexible data center network," in *Proc. of the ACM SIGCOMM*, 2009.

[13] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang,and S. Wright. Power Awareness in Network Design andRouting. In INFOCOM, pages 457 - 465, 2008.

[14] G. Wei, A. Vasilakos, Y. Zheng and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," in *Journal of Supercomput*. 54(2), pp 252-269, Nov. 2010.

[15] A. Gandhi, M. Harchol-Balter, R. Das and C. Lefurgy. "Optimal power allocation in server farms," in *Proc. of intern. joint conf. on measurement and modeling of computer systems*, pp157-168, 2009.

[16] Intel® Xeon® Processor 5600 Series Product Brief: http://www.intel.com/content/www/us/en/processors/xeon/xeon-5600-brief.html

[17] Z. Wang, X. Zhu, C. McCarthy, P. Ranganathan and V. Talwar, "Feedback Control Algorithms for Power Management of Servers," in *Proc. Intern. Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, Jun. 2008.