Energy Efficient Offloading for Competing Users on a Shared Communication Channel

ENERGY EFFICIENT OFFLOADING FOR COMPETING USERS ON A SHARED COMMUNICATION CHANNEL

BY

ERFAN MESKAR, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2015) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Energy Efficient Offloading for Competing Users on a Shared Communication Channel
AUTHOR:	Erfan Meskar B.Sc., (Electrical Engineering) Amirkabir University of Technology, Tehran, Iran
SUPERVISOR:	Dr. Terrence Todd Dr. George Karakostas
NUMBER OF PAGES:	xii, 50

To my beloved mother

Parvaneh

Abstract

In this thesis we consider a set of mobile users that employ cloud-based *computation offloading*. In computation offloading, user energy consumption can be decreased by uploading and executing jobs on a remote server, rather than processing the jobs locally. In order to execute jobs in the cloud however, the user uploads must occur over a base station channel which 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 remotely processed, and may constrain the users ability to reduce energy usage. 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 thesis shows that for a variety of parameters, a game where each user independently sets its offloading decisions always has a pure Nash equilibrium, and a Gauss-Seidel method for determining this equilibrium is introduced. Results are presented which illustrate that the system always converges to a Nash equilibrium using the Gauss-Seidel method. Data is also presented which show the number of Nash equilibria that are found, 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.

Acknowledgements

Foremost, I would like to express my sincere appreciation to my supervisors, Dr. Terence Todd and Dr. George Karakostas, for their patience, encouragement and support through my masters at McMaster University. I am truly grateful for working under their supervision. I would also like to thank Dr. Dongmei Zhao, for her thoughtful advice and insightful criticism through the duration of my studies.

I would like to record my warmest gratitude to my mother since her love and endless support have made it possible for me to excel in my studies.

My kind regards to my fellow colleagues in the Wireless Networking Laboratory, who I have had the pleasure of working with. Special thanks to my friends and lab mates, Hadi Meshgi and Naby Nikookaran for their intellectual supports and suggestions.

Last but not the least, I am particularly grateful to my friends in Hamilton for all the fun and unforgettable moments we had together.

Notations

D_m	required CPU cycles for the m_{th} user
B_m	input bits for the m_{th} user
v_m^l	local energy consumption for the m_{th} user (joules/CPU cycle)
f_m^l	local computation power for the m_{th} user (CPU cycles/second)
f^s	cloud server computation power (CPU cycles/second)
T_m^l	local execution response time for the m_{th} user (seconds)
E_m^l	local execution energy consumption for the m_{th} user (joules)
P_m^t	transmission power for the m_{th} user (watts)
P_m^w	waiting power for the m_{th} user (watts)
r_m	channel data rate for the m_{th} user (bps)
T_m^{off}	offloading time delay for the m_{th} user (seconds)
E_m^{off}	offloading energy consumption for the m_{th} user (joules)
T_m^s	server execution time delay for the m_{th} user (seconds)
T_m^r	total remote execution response time for the m_{th} user (seconds)
E_m^r	total remote execution energy consumption for the m_{th} user (joules)
T_m	total response time for the m_{th} user (seconds)

E_m	total energy consumption for the m_{th} user (joules)
$T_m^{\max}$	maximum tolerable response time for the m_{th} user (seconds)
β_m	exclusive data uploading time for the m_{th} user (seconds)
τ_m	maximum tolerable data uploading time for the m_{th} user (seconds)
Φ_m	negative uploading time margin for the m_{th} user (seconds)
U_m	m_{th} user
n	number of users

Abbreviations

AP	Access Point
API	Application Programing Interface
EC	Elastic Cloud
IaaS	Infrastructure as a Service
MCC	Mobile Cloud Computing
NE	Nash Equilibrium
NEP	Nash Equilibrium Point
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
QoS	Quality of Service
SaaS	Software as a Service
SMD	Smart Mobile Device

Contents

List of Figures

Chapter 1

Introduction and Motivation

The market of the mobile phone has expanded rapidly. With a rapid development of embedded systems and high-speed wireless networks, mobile devices, are increasingly becoming a common stuff of daily human life. By the end of 2009, there were approximately 4.6 billion worldwide mobile cellular subscriptions; 370 times the 1990 number in less than 20 years. Nowadays, we use mobile devices to do many of our daily jobs that we used to do on our desktops. The rapid development of embedded systems, software, and high-speed wireless networks enables them to make calls and send short messages and emails. Moreover, it makes them capable of sensing the environment and making social contacts, healthcare, and mobile learning (Wei *et al.*, 2013). Users can interact with other devices and social community without any time and space restriction because of the inherent mobility of the mobile devices. The dream of Information at your fingertips anywhere, anytime has become true.

Smartphones have become the main choice of interest for computing platform for many users. Many computational intensive applications such as natural language translators (Balan *et al.*, 2007; Flinn *et al.*, 2002), speech recognizers (Balan *et al.*,

2007; Su and Flinn, 2005), optical character recognizers, image processors (Kristensen and Bouvin, 2008; Porras *et al.*, 2009), online games, and video processing (Chun and Maniatis, 2009) are now accessible on smartphones. However, mobile devices have some inherent deficits, such as their limited battery energy, poor computation power, constrained storage space, and insufficient sensing capacities. These limitations have brought many challenges in quality of service (QoS) insurance, energy management, and security issues for mobile applications. The increase in the energy density of smartphones batteries has not fitted the rise in the power demand of these devices, thus, resulting in a power crisis in the smartphone technology development.

A 2005 study in 15 countries revealed that users considered longer battery lifetime more significant than any other features such as storage capacity or cameras (Kumar and Lu, 2010). A survey conducted in 2009 revealed that short battery life is the most undesirable characteristic of Apple iPhone (Paczkowski, 2009).

There are four basic approaches to saving energy and extending battery lifetime in mobile devices (Kumar and Lu, 2010). The first one is to adopt a new generation of semiconductor technology. By shrinking the size of transistors, their energy consumption reduces. However, providing more functionality and better performance necessitates more transistors. Consequently, power consumption may increase. The second one is to keep the whole or individual components of the system in standby or sleep modes to save power. Doubling the processor clock speed approximately octuples the power consumption. Hence, to execute programs more slowly is the third approach to saving energy. However, it may lead to users' dissatisfaction. The last approach that we focus on this thesis is to eliminate computation completely. Rather than executing an application on the mobile device, computation is offloaded

to somewhere else, e.g. a cloud server, to extend mobile system battery lifetime.

1.1 Cloud Computing

Cloud computing is the new computing paradigm in which users have access to the resources as a service over the internet. These resources could be ranging from physical infrastructure resources such as memory and storage to complex processing software. Industry and academics have widely recognized cloud computing to have the potential to transform large parts of the information technology sector. According to ABI Research, more than 240 million business customers will be leveraging cloud computing services through mobile devices, driving revenues of $5.2billions by 2015.

Unfortunately, the definition of the term "cloud computing" is not clear which is not surprising when the term covers a remarkably wide area. The National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. These resources (e.g., networks, servers, storage, applications, and services) can be rapidly provisioned and released with minimal management effort or service provider interaction" (Mell and Grance, 2011).

In recent years, several online file storage services have been presented for augmenting storage potentials of clients such as Amazon S3, Google Docs, MobileMe, and DropBox. Moreover, Amazon web services perform computations using Elastic Cloud Computing (EC2) to alleviate computation restriction of clients.

Cloud service providers use different models for the delivery of cloud resources; such as Software as Service (SaaS), Infrastructure as Service (IaaS), and Platform as a Service (PaaS). A SaaS cloud provides on-demand applications over the Internet,

such as word processing and media streaming. Google Apps, iCloud, and Microsoft Office 365 are good examples of SaaS clouds. A prime example of PaaS cloud is Google App Engine, where cloud computing services are provided as a computing platform using a published Python based application programming interface (API), and Google Inc that developers can use to develop cloud-based applications. With IaaS, resources are leased to tenants in the form of Virtual Machines (such as Amazon EC2, Microsoft Azure, and RackSpace Cloud).

1.2 Mobile Cloud Computing

The lack of resources on smartphones motivates the researchers in mobile computing to search for the infrastructure that can provide the needed resources for the mobile devices (Yang *et al.*, 2012). If a mobile user wants to use a computation intensive application, the computation can be performed in the cloud resources provided by distant data centers, other mobile devices, and network facilities to meet users requirements (such as QoS, security and privacy). Mobile cloud computing (MCC) is evolving as a new computing paradigm that intends to augment resource-constrained mobile devices, taking advantage of the abundant resources hosted by clouds. It brings powerful resources of the cloud centers for mobile devices and applications while having low cost, high scalability, and inheriting the robustness of cloud computing (Wei *et al.*, 2013). Theoretically, Cloud computing can supply mobile devices with inexhaustible resources. The objective of MCC is to mitigate computing resources limitations in mobile devices by employing resources and services of conventional computational clouds. Mobile devices can improve the constrained storage space and processing capabilities by utilizing cloud storage (such as Amazon S3 and

Dropbox) and processing services (Zhang *et al.*, 2011).

Similar to cloud computing, the definition of the term "mobile cloud computing" is not very clear. Aepona defines MCC as a new mobile computing paradigm whereby the storage and the data processing are migrated from the mobile devices to resources rich centralized computing data centers in computational clouds (Alzahrani *et al.*, 2014).

The processing resources on the cloud have much higher speed and larger computation capability. Thus, MCC is becoming an increasingly attractive way to boost the performance of mobile devices by offloading the tasks onto the cloud and carefully scheduling task executions on both the mobile device and the cloud (Liu *et al.*, 2013). However, the task precedence requirements need to be considered to achieve seamless and transparent migration. Applications such as natural language processing, object/gesture recognition, and image/video editing are good candidates for remote cloud execution (Lin *et al.*, 2014). Large applications can be partitioned into various tasks with task-precedence requirements, and the partition-level offloading of computation-intensive tasks prolongs the battery operation time as well as improves the performance by relieving the burden in CPU (Kremer *et al.*, 2003). Hence, mobile devices can support more sophisticated and richer applications and services.

1.3 Motivation

Most of earlier works in this area focused on energy efficiency in single-user mobile cloud computing frameworks. In order to execute jobs in the cloud however, the user uploads must occur over a base station channel that is shared by other uploading users. Congestion on the channel introduces job execution latency and increases the

data uploading energy consumption. Hence, earlier frameworks may not be efficient anymore. A few researchers addressed this problem by focusing on the joint allocation of radio and computational capabilities in multiuser mobile cloud computing (Barbarossa *et al.*, 2013)(Sardellitti *et al.*, 2014). These models proposed a central decision-making unit, which determines users' offloading decisions regarding social utility minimization.

We consider a set of mobile users that access cloud services over a shared base station communication channel. In our model the utility is defined as user's energy consumption. Unlike models proposed by Ge *et al.* (2012), Chen (2015) and Chen *et al.* (2015) which do not refer to job deadlines, the job completion times in our model are subject to hard deadline constraints that may vary from user to user. Users intend to minimize their utility function (energy consumption) while not violating their specified job completion time constraints. 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. Lack of central coordinator radically changes the problem. Selfish users compete for a common resource (the channel) while they are trying to minimize their utility (energy consumption). Researchers model such setting as a game (Ge *et al.*, 2012) (Chen, 2015) (Chen *et al.*, 2015).

Game theory is a useful framework for designing decentralized mechanisms, such

that the mobile device users in the system can self-organize into the mutually satisfactory computation offloading decisions. Moreover, as different mobile devices are owned by different individuals and they may pursue different interests, game theory is a powerful tool to analyze the interactions among multiple mobile device users who act in their own interests. Each user chooses the best strategy (local execution or remote execution) in order to minimize its utility function. A user's utility is a function of its own and other users' strategies. For example, remote execution time and offloading energy consumption will increase if more users choose remote execution and upload their data on the shared communication channel. Consequently, a user's best strategy is affected by other users' strategies. Users play the game until they reach a stable state where no one would benefit by defecting, i.e., a Nash Equilibrium (NE).

Users 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 users play the game 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. 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).

The rest of the thesis is organized as follows. Chapter 2 is the related work section. In Chapter 3 we give a detailed description of the system model and formulate the central decision-making unit. In Chapter 4 we assume that the users are selfish and model the system as a game. Then we prove that this game always has a pure Nash equilibrium and propose an algorithm to find it. Performance results are given in Chapter 5. Finally, Chapter 6 contains the conclusions of our work and possible future research topics.

Chapter 2

Literature Review

2.1 Architectures

Figure 2.1 shows the three major components of the generic MCC model (i.e. SMDs, wireless internet technology, and computational cloud). In order to access the computational cloud, SMDs use various wireless network technology technologies (e.g., 3G, LTE, or Wi-Fi) that are inherently less reliable than their wired counterparts due to mobility requirements (Shiraz *et al.*, 2013). Researchers have proposed various architectures for MCC, such as MobiCloud, MAUI, CloneCloud, Cloudlet, and Hyrax to support different types of applications (Wei *et al.*, 2013). We divide these architectures into two main categories depending on where the users execute their tasks and how mobile devices and the cloud can be connected.

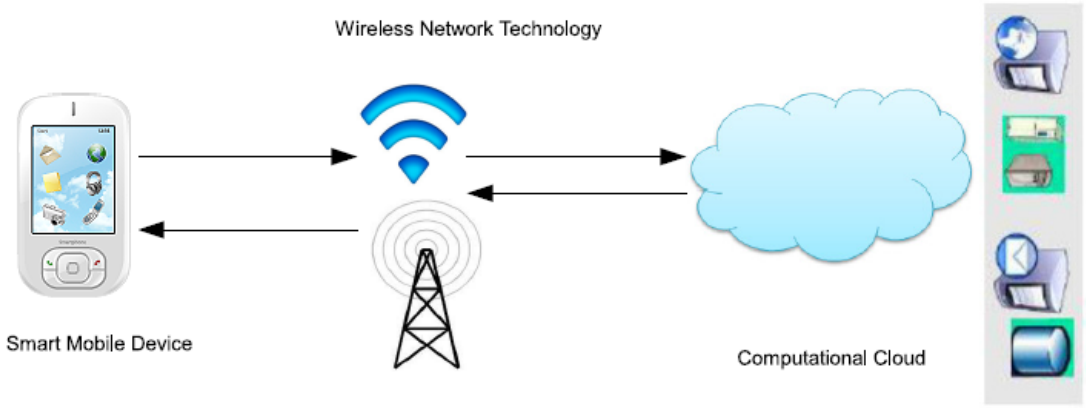

Figure 2.1: Model of Mobile Cloud Computing

2.1.1 Agent-client Scheme

In the agent-client scheme, mobile users offload their tasks to a remote data center in the cloud. This data center can be located in a large distant cloud center (e.g., Amazon and Google cloud center) or a local service infrastructure implemented near the access point of the mobile device connection (Cloudlet).

Figure 2.1 shows a typical model of MCC system with a distant cloud (such as MAUI (Cuervo *et al.*, 2010) and CloneCloud (Chun and Maniatis, 2009)), where mobile devices offload applications to the cloud via wireless networks. When clouds finish the computation tasks, the results are transmitted back to the mobile users. Satyanarayanan proposed Cloudlet as a local cloud implemented at the access point of the mobile users (Satyanarayanan *et al.*, 2009; Dinh *et al.*, 2013). Figure 2.2 shows an MCC system with a Cloudlet, whose computation capability is typically less powerful than an enterprise grade cloud server. This arrangement, however, can decrease server response time since mobile devices can connect to the nearby local cloud by using a fast connection such as WiFi. Whenever the computation resources in the cloudlet

Figure 2.2: Cloudlet

do not satisfy user's expectations, it can access the enterprise distant cloud servers to utilize more powerful resources.

2.1.2 Collaborative Scheme

As in Mobile Peer-to-Peer systems and mobile grid computing, mobile devices collaborate in order to utilize their free resources to run applications (such as Hyrax (Marinelli, 2009), Misco (Dou *et al.*, 2010), and the virtual cloud (Huerta-Canepa and Lee, 2010)). The cloud server in these schemes function as the controller and scheduler for collaboration (Guan *et al.*, 2011). Figure 2.3 shows the general architecture of this type of collaboration system.

In Hyrax, for example, Android smartphones can utilize data computational resources on heterogeneous networks of smartphones and servers.

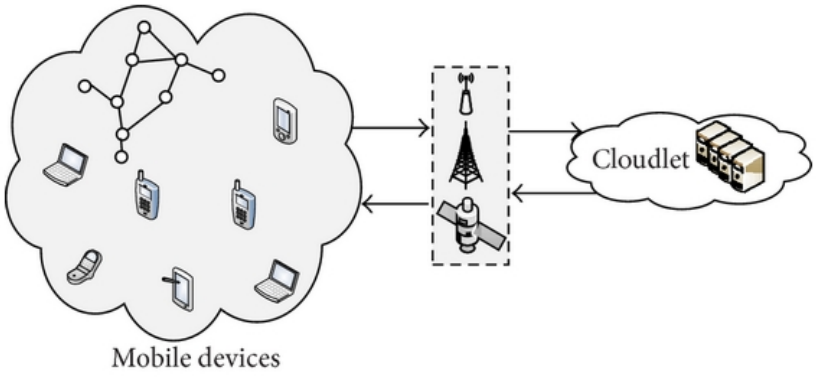

Figure 2.3: The General Architecture of Collaboration Scheme

2.2 Challenges

2.2.1 Network Transmission Between Cloud and Mobile Users

MCC relies on the availability of the cloud and reliable end-to-end communication. Unfortunately, this is often influenced by uncontrollable factors, such as the intermittency of wireless communications (Wu and Wolter, 2014). User mobility may require that they handoff between basestations or access points, leading to undesirable behaviour (Guan *et al.*, 2011). One of the fundamental obstacles in MCC is excessive wireless network latency (Satyanarayanan *et al.*, 2009). Minimizing the use of network can significantly save energy (Dinh *et al.*, 2013) (Baliga *et al.*, 2011).

Network latency and limited bandwidth in the mobile network

The inherent nature of mobile devices and wireless networks raise the problem of network latency in mobile cloud computing (Guan *et al.*, 2011). Mobile applications such as online gaming and speech recognition, which need to be processed in the cloud to conserve energy (Barbera *et al.*, 2013), are very sensitive to execution time.

Excessive network latency may prohibit the use of MCC in such mobile applications. Hence, solutions are required to that will monitor and improve network conditions so that energy can be saved by task offloading while guaranteeing a satisfying user experience.

Galinina *et al.* (2013) proposed a power control scheme that maximizes the energy efficiency of a mobile device transmitting on several communication channels while ensuring the required minimum quality of service. They obtain the optimal transmit power from the solution of an optimization problem that is based on Shannon's capacity formula.

Delayed Offloading

Recently, "delayed" offloading has been proposed to tackle the problem of temporary WiFi network unavailability. When there is no WiFi available, traffic can be delayed up to some chosen deadline (Mehmeti and Spyropoulos, 2013), since delay-tolerant applications are less sensitive to network delays. In participatory sensing applications, for example, data is uploaded from a smartphone to a back-end cloud server either through cellular or WiFi connectivity (Ra *et al.*, 2010). Some of the sensor data are not time-critical, and it is possible to delay transfers until a lower energy WiFi connection is available.

Some work has considered energy-efficient delayed offloading by monitoring the network connectivity (e.g., 3G, WiFi) and exploring the trade of between execution time and transmit power of different network interfaces (Shu *et al.*, 2013). Reference Ra *et al.* (2010) presented a principled approach for designing an optimal online algorithm, called SALSA, for this energy-delay tradeoff using the Lyapunov optimization

framework.

Network Selection

Nowadays, mobile devices are often equipped with multiple wireless networking technologies, such as 3G/EDGE, 4G LTE, and WiFi. Selecting the best available link can reduce energy consumption significantly since the energy-efficiency and availability of these networks can vary significantly. Connectivity may change from place to place, and the data transmission bandwidth (i.e. the uplink and downlink) fluctuates due to multiple factors (such as weather, mobility, and channel fading) (Shu *et al.*, 2013). For these reasons, algorithms are required that choose an appropriate network interface to minimize energy consumption.

Rahmati and Zhong (2007) proposed an online wireless network selection mechanism that trades off offloading transmission efficiency on an intermittently available WiFi network, and the energy consumption needed to search of a better WiFi connection. By monitoring the dynamics of user traffic and energy aware allocation of the radio network resource to mobile users, Gribaudo *et al.* (2013) developed a framework based on a Markovian agent formalism to minimize user energy consumption.

2.2.2 Partitioning and Offloading between the Cloud and Mobile User

Research has proposed partitioning frameworks that divide complex applications into components that can be offloaded separately. However, the data dependency of these components must be considered in order to achieve a smooth application execution (Zhu *et al.*, 2013). This requires the estimation of the communication and

computation energy consumption (i.e. local execution energy consumption) of each component (Kumar and Lu, 2010). Unfortunately, the problem of optimal application partitioning is NP-complete and therefore heuristic techniques are required (Gu *et al.*, 2004). These methods are classified as static or dynamic partitioning. Static partitioning cannot fit all application scenarios since it is only suitable for one specific application (Chun and Maniatis, 2010). Dynamic partitioning, however, can addresses various MCC environments (e.g., different device platforms, networks, and clouds), and diverse applications and dynamic application processing loads (Zhu *et al.*, 2013).

Kumar et. al. formulated a simple optimization problem that minimizes mobile user energy consumption; where the partitioning decision is derived from the direct solution of the optimization (Kumar and Lu, 2010). Cuervo *et al.* (2010) proposed MAUI to maximize the energy benefits of offloading code while utilizing "managed coding" to reduce the burden on programmers to deal with program partitioning. Giurgiu *et al.* (2009) presented ALL and K-step, which are static and dynamic two-step partitioning approaches, respectively. First, the application's components are modeled as a data flow graph. In the second step, an optimization algorithm finds the optimum partitioning decision to maximize user utility.

Chapter 3

System Model and Problem Formulation

3.1 Overview

In this chapter we describe the MCC system that our research is based on. Our model is based on cloudlet system which was proposed by Satyanarayanan to eliminate the long distance between mobile users and cloud servers. Satyanarayanan, for instance, proposed a cloudlet system in popular public coffeehouses in which a lot of customers use their mobile devices(such as laptop, tablet, and smartphone) everyday. By using close-by servers located at wireless access point (AP) which are at just one-hop distance from mobile users. The closer the users are to the server, the faster they can transmit data. Moreover, less energy is consumed on data transmission so users decide to offloade their computation more probably.

3.2 System Model

The system considered is shown in Figure 3.1. 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 which is needed to run the job on the remote server. When a set of stations 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.

Assumption 1. *During each visit, the scheduler employs a gated discipline: it processes all jobs that are present at the time of the visit. Hence, we can assume that all jobs arrive into the system simultaneously.*

Assumption 2. *There is an unlimited number of cloud servers.*

Assumption 3. *The uploading bit rates for each user may be different due to differing radio propagation path loss values, and therefore the user data payload depends on which user is currently transmitting.*

Assumption 4. *The upload bit rates are constant for the duration of a given job contention/uploading cycle.*

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

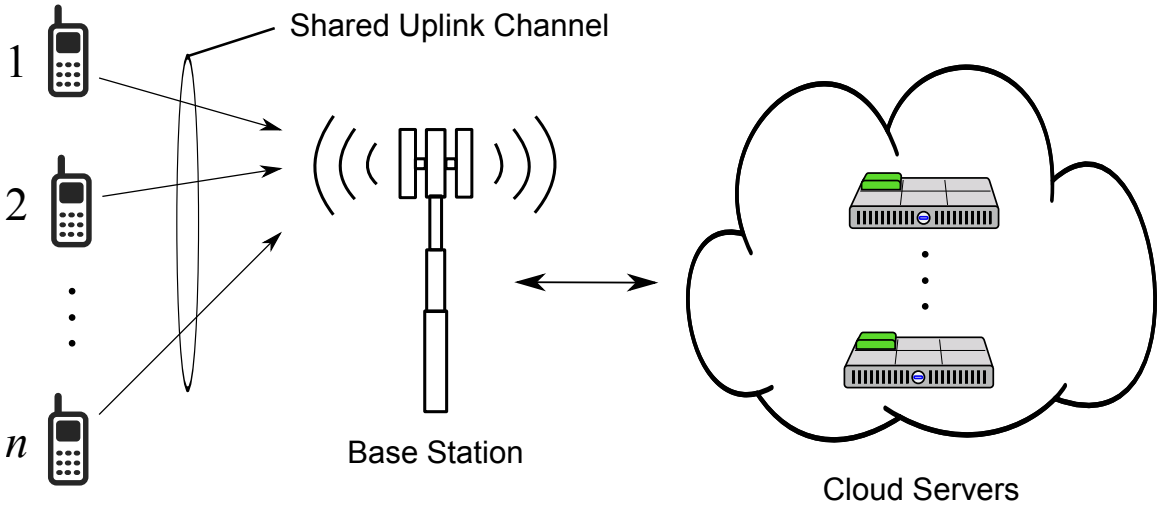

Figure 3.1: Mobile Computation Offloading Model. n mobile users access infrastructure-based cloud servers over a shared wireless communication channel.

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 3.1 summarizes the notation used.

User U_m, for $m \in \{1, 2, \ldots, n\}$ 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 $\tau_m = T_m^{\max} - \frac{D_m}{f_m^l}$.

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.

3.2.1 Local Processing

In the case where user U_m decides to execute its job locally, we use the simple model described in Kumar and Lu (2010) where the local execution energy consumption E_m^l

Table 3.1: 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^{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

and the time delay due to local computation T_m^l are defined as follows:

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

3.2.2 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_m^{off}. 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

$$T_{m+1}^{\text{off}} = T_m^{\text{off}} + (\beta_{m+1} - \beta_m)\, \eta_{m+1} \tag{3.1}$$

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, (3.1) implies for an uploading user U_m (i.e., $a_m = 1$), that

$$T_m^{\text{off}} = \begin{cases} (1 + \sum_{i=m+1}^{n} a_i)\beta_m & \text{if } m = 1 \\ \sum_{i=1}^{m-1} a_i\beta_i + (1 + \sum_{i=m+1}^{n} a_i)\beta_m & \text{if } 1 < m < n \\ \sum_{i=1}^{m-1} a_i\beta_i + \beta_m & \text{if } m = n \end{cases} \tag{3.2}$$

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

$$E_m^{\text{off}} = P_m^t \beta_m + P_m^w (T_m^{\text{off}} - \beta_m) \tag{3.3}$$

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

$$T_m^s = \frac{D_m}{f^s} \tag{3.4}$$

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

$$T_m^r = T_m^{\text{off}} + T_m^s \tag{3.5}$$

$$E_m^r = E_m^{\text{off}} = P_m^t \beta_m + P_m^w (T_m^{\text{off}} - \beta_m) \tag{3.6}$$

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

$$T_m = a_m T_m^r + (1 - a_m) T_m^l \tag{3.7}$$

$$E_m = a_m E_m^r + (1 - a_m) E_m^l \tag{3.8}$$

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.

3.3 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:

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

Using (3.2), (3.6) and (3.8), the objective function of (OPT_SUM) can be written as

$$\begin{aligned} \sum_{m=1}^{n} E_m = & \sum_{m=1}^{n} (P_m^t - P_m^w)\beta_m a_m + \sum_{m=1}^{n} (1-a_m) v_m^l D_m \\ & + \sum_{m=1}^{n} a_m P_m^w \Big(\sum_{i<m} a_i \beta_i + \beta_m \sum_{i>m} a_i\Big) \end{aligned} \tag{3.9}$$

Chapter 4

Selfish Decision Making

4.1 Overview

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 3.3 is to just provide the agents with channel information. As a result, we adopt a game theoretic approach in order to study our setting.

4.2 Game Theoretic Model

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:

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

Note that (3.7) and (3.8) 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 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 (Koutsoupias and Papadimitriou, 1999) 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 (Koutsoupias and Papadimitriou, 1999) 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.

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). If one allows the game dynamics to run their course by playing the game in iterations and in a distributed way, the number of iterations may be prohibitively large, or (even worse) they may never converge to an equilibrium. That is why we propose the replacement of this actual playing of the game among the users with the simulation of the game by a single machine (the controller) that can calculate the equilibrium strategies without a single iteration of the game being played. 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,

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
```

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.

4.3 Nash Equilibrium Existence

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

$$\tau_m = T_m^{\max} - \frac{D_m}{f^s}, \tag{4.1}$$

then we can rewrite (mOPT) as

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

Given Φ_m as follows

$$\Phi_m = T_m^{\text{off}} - \min\{\tau_m, \frac{v_m^l D_m}{P_m^w} - (\frac{P_m^t}{P_m^w} - 1)\frac{B_m}{r_m}\} \tag{4.2}$$

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 (4.2)

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

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 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.*

Algorithm 2 Finding Nash equilibrium in hetrogeneous system

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

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 ($1 \leq k \leq n$), 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 $0 \leq k \leq m$, 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} ($1 \leq i \leq m$) 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 ($1 \leq i \leq m$) have entered S before iteration $m + 1$. In iteration i, U_{ρ_i} ($1 \leq i \leq m + 1$) 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,

$$\Phi_{\rho_i}(\vec{\mathbf{x}}_{\rho_i}) > 0,\ 1 \leq i \leq m \tag{4.3}$$

Obviously, we already have that

$$\Phi_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) > 0. \tag{4.4}$$

We show (4.3) 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)

$$\Phi_{\rho_m}(\vec{\mathbf{a}}) > \Phi_{\rho_{m+1}}(\vec{\mathbf{a}}) \tag{4.5}$$

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 (3.2) shows that

$$T^{\text{off}}_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) = T^{\text{off}}_{\rho_m}(\vec{\mathbf{a}}) - \beta_{\rho_m} \Rightarrow \Phi_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) = \Phi_{\rho_m}(\vec{\mathbf{a}}) - \beta_{\rho_m} \tag{4.6}$$

$$T^{\text{off}}_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) = T^{\text{off}}_{\rho_{m+1}}(\vec{\mathbf{a}}) - \beta_{\rho_m} \Rightarrow \Phi_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) = \Phi_{\rho_{m+1}}(\vec{\mathbf{a}}) - \beta_{\rho_m} \tag{4.7}$$

Then equations (4.5), (4.4), (4.6), and (4.7) 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 (3.2) we would have

$$T^{\text{off}}_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) = T^{\text{off}}_{\rho_m}(\vec{\mathbf{a}}) - \beta_{\rho_m} \Rightarrow \Phi_{\rho_m}(\vec{\mathbf{x}}_{\rho_m}) = \Phi_{\rho_m}(\vec{\mathbf{a}}) - \beta_{\rho_{m+1}} \tag{4.8}$$

$$T^{\text{off}}_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) = T^{\text{off}}_{\rho_{m+1}}(\vec{\mathbf{a}}) - \beta_{\rho_m} \Rightarrow \Phi_{\rho_{m+1}}(\vec{\mathbf{x}}_{\rho_{m+1}}) = \Phi_{\rho_{m+1}}(\vec{\mathbf{a}}) - \beta_{\rho_{m+1}} \tag{4.9}$$

Then equations (4.5), (4.4), (4.8), and 4.9 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 (4.3) 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

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{a}}) \geq \Phi_{\rho_z}(\vec{\mathbf{a}}) \tag{4.10}$$

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 (3.2) implies that

$$T^{\text{off}}_{\rho_{l-1}}(\vec{\mathbf{b}}) = T^{\text{off}}_{\rho_{l-1}}(\vec{\mathbf{a}}) - \sum_{j\in S_L} \beta_j \Rightarrow \Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) = \Phi_{\rho_{l-1}}(\vec{\mathbf{a}}) - \sum_{j\in S_L} \beta_j \tag{4.11}$$

$$T^{\text{off}}_{\rho_z}(\vec{\mathbf{b}}) = T^{\text{off}}_{\rho_z}(\vec{\mathbf{a}}) - \sum_{j\in S_L} \beta_j \Rightarrow \Phi_{\rho_z}(\vec{\mathbf{b}}) = \Phi_{\rho_z}(\vec{\mathbf{a}}) - \sum_{j\in S_L} \beta_j \tag{4.12}$$

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 (4.10), (4.11), and (4.12) show that

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) \geq \Phi_{\rho_z}(\vec{\mathbf{b}}) \tag{4.13}$$

Equation (3.2) also implies that

$$T^{\text{off}}_{\rho_{l-1}}(\vec{\mathbf{c}}) = T^{\text{off}}_{\rho_{l-1}}(\vec{\mathbf{b}}) - |S_U|\beta_{\rho_{l-1}} \Rightarrow \Phi_{\rho_{l-1}}(\vec{\mathbf{c}}) = \Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) - |S_U|\beta_{\rho_{l-1}} \tag{4.14}$$

$$T^{\text{off}}_{\rho_z}(\vec{\mathbf{c}}) = T^{\text{off}}_{\rho_z}(\vec{\mathbf{b}}) - |S_U|\beta_{\rho_z} \Rightarrow \Phi_{\rho_z}(\vec{\mathbf{c}}) = \Phi_{\rho_z}(\vec{\mathbf{b}}) - |S_U|\beta_{\rho_z} \tag{4.15}$$

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 (4.13), (4.14), and (4.15), implies that

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{c}}) \geq \Phi_{\rho_z}(\vec{\mathbf{c}}) \tag{4.16}$$

Then (3.2) implies that

$$T^{\text{off}}_{\rho_{l-1}}(\vec{\mathbf{x}}_{\rho_{l-1}}) = T^{\text{off}}_{\rho_{l-1}}(\vec{\mathbf{c}}) - \beta_{\rho_{l-1}} \Rightarrow \Phi_{\rho_{l-1}}(\vec{\mathbf{x}}_{\rho_{l-1}}) = \Phi_{\rho_{l-1}}(\vec{\mathbf{c}}) - \beta_{\rho_{l-1}} \tag{4.17}$$

$$T^{\text{off}}_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) = T^{\text{off}}_{\rho_z}(\vec{\mathbf{c}}) - \beta_{\rho_{l-1}} \Rightarrow \Phi_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) = \Phi_{\rho_z}(\vec{\mathbf{c}}) - \beta_{\rho_{l-1}} \tag{4.18}$$

and, therfore, from the inductive hypothesis and equations (4.16), (4.17), and (4.18), 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.

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.

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

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{a}}) \geq \Phi_{\rho_z}(\vec{\mathbf{a}}) \tag{4.19}$$

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 (3.2) implies the same equations as (4.11) and (4.12) (recall that $S_U = \emptyset$ in $\vec{\mathbf{b}}$), and therefore we get that

$$\Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) \geq \Phi_{\rho_z}(\vec{\mathbf{b}}). \tag{4.20}$$

We also have

$$T^{\text{off}}_{\rho_{l-1}}(\vec{\mathbf{x}}_{l-1}) = T^{\text{off}}_{\rho_{l-1}}(\vec{\mathbf{b}}) - \beta_{\rho_z} \Rightarrow \Phi_{\rho_{l-1}}(\vec{\mathbf{x}}_{l-1}) = \Phi_{\rho_{l-1}}(\vec{\mathbf{b}}) - \beta_{\rho_z} \tag{4.21}$$

$$T^{\text{off}}_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) = T^{\text{off}}_{\rho_z}(\vec{\mathbf{b}}) - \beta_{\rho_z} \Rightarrow \Phi_{\rho_z}(\vec{\mathbf{x}}_{\rho_z}) = \Phi_{\rho_z}(\vec{\mathbf{b}}) - \beta_{\rho_z} \tag{4.22}$$

Equations (4.20), (4.21), and (4.22) 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. □

Chapter 5

Performance Evaluation

5.1 System Parameters

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, D_m, B_m, r_m, and P_m^t were generated using a random uniform distribution. The required CPU cycles, D_m, were chosen randomly between 1 and 10 Gcycles. Input data size, B_m, was between 0.42 to 42 Mb and the channel data rate, r_m, ranged from 6.4 to 64 Mbps. Data transmission power, P_m^t, was between 0.75 to 1 mW. Local computation power, f_m^l, was selected randomly from 0.5, 0.8 or 1 giga CPU cycles/sec and local execution power consumption, P_m^l, was chosen randomly from 20 , 22.5 and 25 mW with equal probability. Cloud server computation power, f^s, was taken to be 100 giga CPU cycles/sec and local energy consumption, v_m^l, was considered to be equal to P_m^l/f_m^l.

5.2 Convergence Time

The convergence time of Algorithm 2 and the Gauss-Seidel algorithm was studied and the results are shown in Figure 5.1. Algorithm 2 has order of n^2 time complexity. The two dashed lines show the maximum convergence time among all reached NEPs for each 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 however, was not observed in any of our experiments, i.e., Gauss-Seidel always found a Nash equilibrium. 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.

5.3 Offloading Ratio

Table 5.1 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

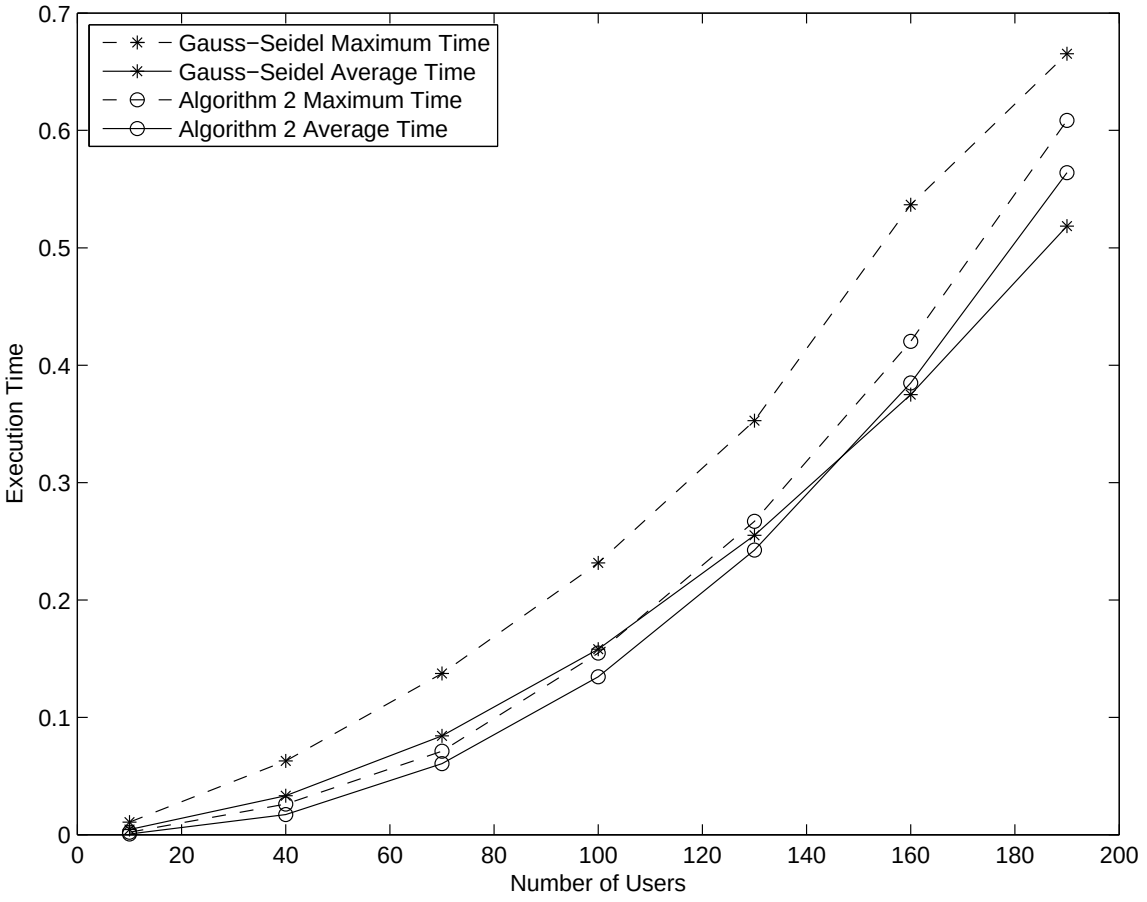

Figure 5.1: Convergence Time vs. Number of Users

becomes prohibitively large for an ever greater proportion of users. Consequently, the MCC approach is more beneficial in small to moderate size groups.

5.4 Energy Performance

In Figure 5.2, 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

Table 5.1: 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 decrease by increasing the number of users since the offloading ratio becomes smaller.

Figure 5.3 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). Our simulation shows that the PoA as defined by Koutsoupias and Papadimitriou (1999) could be approximately 3. However, proof on theoretical upper bounds for the worst-case equilibrium ratio is required. 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,

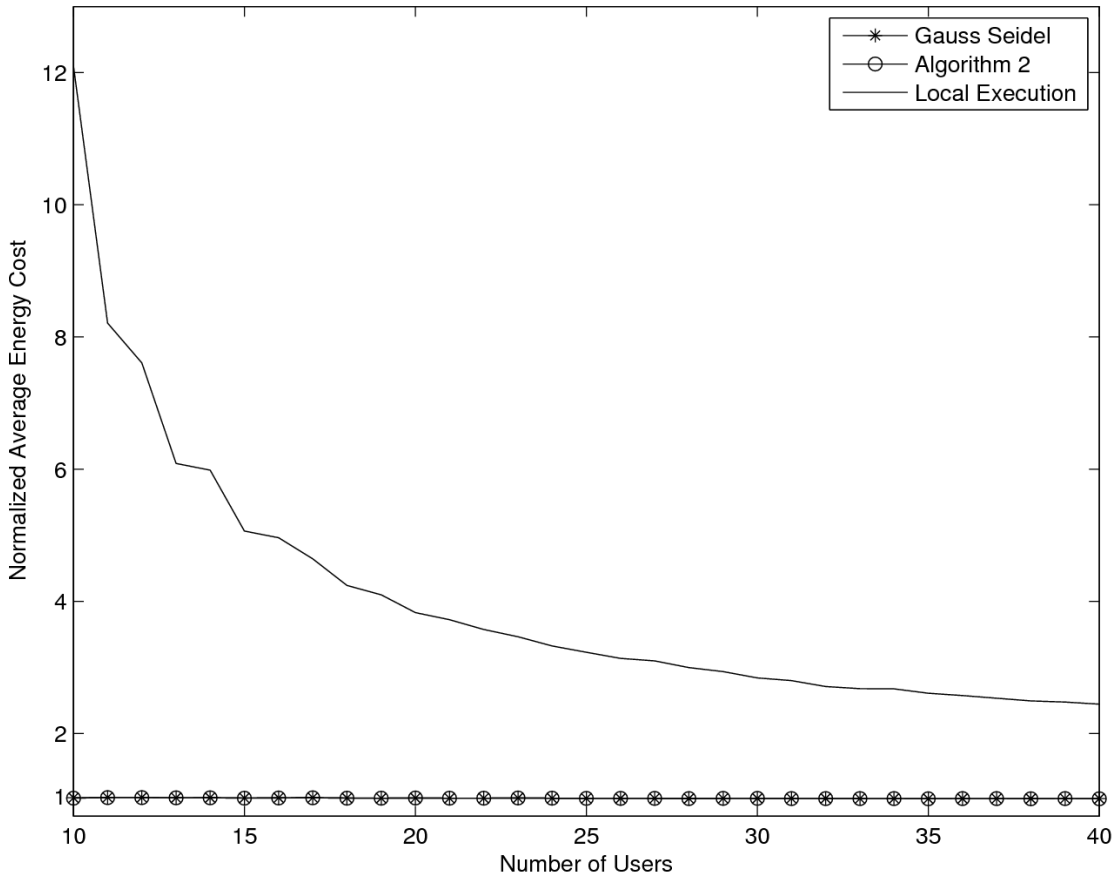

Figure 5.2: Normalized Energy Cost versus Number of Users for Algorithm 2, Gauss-Seidel Algorithm and Local Execution

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 Koutsoupias and Papadimitriou (1999)).

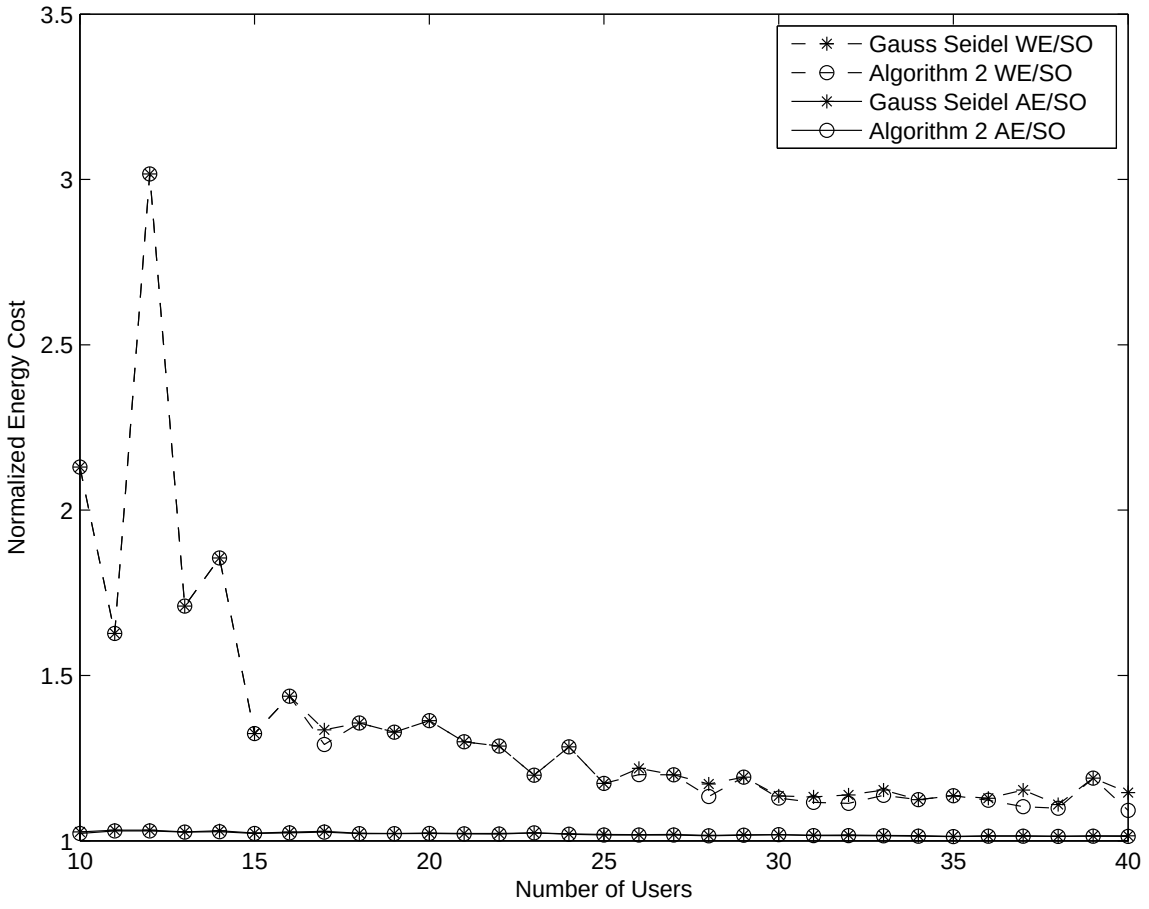

Figure 5.3: Normalized Social Energy Consumption Over All Discovered NEs

Chapter 6

Conclusion

In this thesis 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 other 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 thesis showed that, 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 iterations required, and the quality of the equilibria 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 found that the solutions perform well compared

to a lower bound on total energy performance.

In this thesis, we assume that the users and the controller are truthful in reporting data to each other. The problem of untruthful users or controller that report misleading information is left open for future work. We may be able to increase the fairness of the reached Nash equilibrium by manipulating the information that the central broker sends back to the mobile users. Wei *et al.* (2010) proposed a method to guarantee fairness of NEPs in their cloud computing model.

We left open the question of theoretical upper bounds for the worst-case equilibrium ratio (i.e., the PoA as defined by Koutsoupias and Papadimitriou (1999)) and theoretical lower bound for the quality of the equilibria obtained from Algorithm 1. Chen (2015) could theoretically assess the efficiency ratio of the worst-case Nash equilibrium over the optimal centralized solution. However, the proposed upper bound appears to be overly conservative.

The work in this thesis is based on binary offloading decisions (i.e., local execution and entire task offloading). For future work, multiple decisions (i.e. multiple job partitions) may be taken into account, such that users can decide what portion of the application to offload. We can deploy the task graph model presented by Lin *et al.* (2014) in order to benefit from application partitioning in out framework.

On the other hand, in the current work each mobile user has a fixed transmission data rate when uploading data to the cloud server through wireless channels. In the future, we can consider stochastic channel conditions in which case the transmission rates of mobile users are no longer constant.

Bibliography

Alzahrani, A., Alalwan, N., and Sarrab, M. (2014). Mobile cloud computing: Advantage, disadvantage and open challenge. In *Proceedings of the 7th Euro American Conference on Telematics and Information Systems*, EATIS '14, pages 21:1–21:4, New York, NY, USA. ACM.

Balan, R. K., Gergle, D., Satyanarayanan, M., and Herbsleb, J. (2007). Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 272–285. ACM.

Baliga, J., Ayre, R. W., Hinton, K., and Tucker, R. S. (2011). Green cloud computing: Balancing energy in processing, storage, and transport. *Proceedings of the IEEE*, **99**(1), 149–167.

Barbarossa, S., Sardellitti, S., and Di Lorenzo, P. (2013). Joint allocation of computation and communication resources in multiuser mobile cloud computing. In *Signal Processing Advances in Wireless Communications (SPAWC), 2013 IEEE 14th Workshop on*, pages 26–30. IEEE.

Barbera, M., Kosta, S., Mei, A., and Stefa, J. (2013). To offload or not to offload?

the bandwidth and energy costs of mobile cloud computing. In *INFOCOM, 2013 Proceedings IEEE*, pages 1285–1293. IEEE.

Chen, X. (2015). Decentralized computation offloading game for mobile cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, **26**(4), 974–983.

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

Chun, B.-G. and Maniatis, P. (2009). Augmented smartphone applications through clone cloud execution. In *HotOS*, volume 9, pages 8–11.

Chun, B.-G. and Maniatis, P. (2010). Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, page 7. ACM.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. (2010). Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, **13**(18), 1587–1611.

Dou, A., Kalogeraki, V., Gunopulos, D., Mielikainen, T., and Tuulos, V. H. (2010). Misco: a mapreduce framework for mobile systems. In *Proceedings of the 3rd international conference on pervasive technologies related to assistive environments*, page 32. ACM.

Flinn, J., Park, S. Y., and Satyanarayanan, M. (2002). Balancing performance, energy, and quality in pervasive computing. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 217–226. IEEE.

Galinina, O., Trushanin, A., Shumilov, V., Maslennikov, R., Saffer, Z., Andreev, S., and Koucheryavy, Y. (2013). Energy-efficient operation of a mobile user in a multi-tier cellular network. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 198–213. Springer.

Ge, Y., Zhang, Y., Qiu, Q., and Lu, Y.-H. (2012). A game theoretic resource allocation for overall energy minimization in mobile cloud computing system. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 279–284. ACM.

Giurgiu, I., Riva, O., Juric, D., Krivulev, I., and Alonso, G. (2009). Calling the cloud: enabling mobile phones as interfaces to cloud applications. In *Middleware 2009*, pages 83–102. Springer.

Gribaudo, M., Manini, D., and Chiasserini, C. (2013). Studying mobile internet technologies with agent based mean-field models. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 112–126. Springer.

Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., and Milojicic, D. (2004). Adaptive offloading for pervasive computing. *Pervasive Computing, IEEE*, **3**(3), 66–73.

Guan, L., Ke, X., Song, M., and Song, J. (2011). A survey of research on mobile cloud computing. In *2011 10th IEEE/ACIS International Conference on Computer and Information Science*, pages 387–392. IEEE.

Huerta-Canepa, G. and Lee, D. (2010). A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, page 6. ACM.

Koutsoupias, E. and Papadimitriou, C. (1999). Worst-case Equilibria. In *16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 404–413.

Kremer, U., Hicks, J., and Rehg, J. (2003). A compilation framework for power and energy management on mobile computers. In *Languages and Compilers for Parallel Computing*, pages 115–131. Springer.

Kristensen, M. and Bouvin, N. (2008). Developing cyber foraging applications for portable devices. In *Portable Information Devices, 2008 and the 2008 7th IEEE Conference on Polymers and Adhesives in Microelectronics and Photonics. PORTABLE-POLYTRONIC 2008. 2nd IEEE International Interdisciplinary Conference on*, pages 1–6.

Kumar, K. and Lu, Y.-H. (2010). Cloud computing for mobile users: Can offloading computation save energy? *Computer*, **43**(4), 51–56.

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

Liu, F., Shu, P., Jin, H., Ding, L., Yu, J., Niu, D., and Li, B. (2013). Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *Wireless Communications, IEEE*, **20**(3), 14–22.

Marinelli, E. E. (2009). Hyrax: cloud computing on mobile devices using mapreduce. Technical report, DTIC Document.

Mehmeti, F. and Spyropoulos, T. (2013). Performance analysis of "on-the-spot" mobile data offloading. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 1577–1583. IEEE.

Mell, P. M. and Grance, T. (2011). Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States.

Paczkowski, J. (2009). iphone owners would like to replace battery, at&t. Technical report.

Porras, J., Riva, O., and Kristensen, M. D. (2009). Dynamic resource management and cyber foraging. In *Middleware for Network Eccentric and Mobile Applications*, pages 349–368. Springer.

Ra, M.-R., Paek, J., Sharma, A. B., Govindan, R., Krieger, M. H., and Neely, M. J. (2010). Energy-delay tradeoffs in smartphone applications. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 255–270. ACM.

Rahmati, A. and Zhong, L. (2007). Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 165–178. ACM.

Sardellitti, S., Scutari, G., and Barbarossa, S. (2014). Distributed joint optimization

of radio and computational resources for mobile cloud computing. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 211–216. IEEE.

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

Shiraz, M., Gani, A., Khokhar, R. H., and Buyya, R. (2013). A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys & Tutorials, IEEE*, **15**(3), 1294–1313.

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

Su, Y.-Y. and Flinn, J. (2005). Slingshot: Deploying stateful services in wireless hotspots. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, MobiSys '05, pages 79–92, New York, NY, USA. ACM.

Wei, G., Vasilakos, A. V., Zheng, Y., and Xiong, N. (2010). A game-theoretic method of fair resource allocation for cloud computing services. *The journal of supercomputing*, **54**(2), 252–269.

Wei, X., Fan, J., Lu, Z., and Ding, K. (2013). Application scheduling in mobile cloud computing with load balancing. *Journal of Applied Mathematics*, **2013**.

Wu, H. and Wolter, K. (2014). Dynamic transmission scheduling and link selection in mobile cloud computing. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 61–79. Springer.

Yang, D., Xue, G., Fang, X., and Tang, J. (2012). Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 173–184. ACM.

Zhang, X., Kunjithapatham, A., Jeong, S., and Gibbs, S. (2011). Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, **16**(3), 270–284.

Zhu, C., Leung, V., Hu, X., Shu, L., and Yang, L. T. (2013). A review of key issues that concern the feasibility of mobile cloud computing. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 769–776. IEEE.