Energy Fair Cloud Server Scheduling in Mobile Computation Offloading

ENERGY FAIR CLOUD SERVER SCHEDULING IN MOBILE COMPUTATION OFFLOADING

BY

JIANTING YUE, B.Eng.

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 Fair Cloud Server Scheduling in Mobile Computation Offloading
AUTHOR:	Jianting Yue B.Eng., (Information Engineering) Shanghai Jiao Tong University, Shanghai, China
SUPERVISORS:	Dr. Dongmei Zhao and Dr. Terence D. Todd
NUMBER OF PAGES:	xv, 64

This thesis is dedicated to my beloved parents for their unconditional love and support.

Abstract

This thesis considers the issue of energy fairness in mobile *computation offloading*. In computation offloading, mobile users reduce their energy consumption by executing jobs on a remote cloud server, rather than processing the jobs locally. This can result in energy unfairness however, when the processed jobs are subject to hard deadline constraints.

In this thesis we consider how energy unfairness can be compensated for, by smart scheduling at the cloud server. We first derive an optimum offline scheduler using an integer linear program (ILP) that uses a min-max energy objective and preemptive cloud server scheduling. We then introduce several online scheduling algorithms. The first one is referred to as First-Generated-First-Scheduled (FGFS), where jobs that are generated earlier are given cloud server priority. A modified version, referred to as γ-Ratio Accepted FGFS (γ-FGFS) is proposed, where the acceptance of job submission is subject to an energy threshold test. A version of this algorithm, γ-Ratio Accepted Earliest Deadline First (γ-EDF), is considered that uses earliest deadline first (EDF) scheduling to test for job feasibility.

We also include comparisons using an analytic model that shows the performance possible when the system uses optimum open loop job submission (OLJS) with first-come-first-served (FCFS) cloud server scheduling. Various performance results are

presented that show the improvements in energy fairness possible with the proposed schedulers.

Acknowledgements

After two years' work, my Master's study at McMaster University is coming to an end. I would like to express my sincere appreciation to everyone who has provided me with valuable help.

To start with, I would like to express my heart-felt gratitude to my supervisors, Dr. Dongmei Zhao and Dr. Terence D. Todd, for their continuous encouragement, advice and support throughout my entire research work. They have helped me through all the stages of completing this thesis, which could have never happened without their help. I am truly grateful for working under their supervision.

My sincere gratitude particularly goes to Dr. Bo Yang at the Department of Automation in Shanghai Jiao Tong University. It was he who guided me to the magical world of academic research four years ago. He kindly and patiently helped me overcome the difficulties at the very outset of my research experience, and steadily motivated me to become able to work in this field.

My thanks go to my beloved parents for their consideration and confidence in me throughout the past two decades. They have been constantly lending support to me and encouraging me to make progress.

It has been a wonderful experience studying in McMaster University and working in the Wireless Networking Laboratory, where I got to know lots of amiable friends and

intelligent colleagues including Na Yang, Wen Huang, Lei Qin, and Naby Nikookaran.

Finally, I owe my sincere gratitude to all the professors, instructors and friends who have either shared my happiness or helped me get through any difficulty.

Notations

$\mathcal{M}$	Set of mobile user classes
M	Number of mobile user class
$\mathcal{J}_m$	Set of jobs generated by mobile user class m
J_m	Number of jobs generated by mobile user class m
$\mathcal{T}$	Complete set of time slots
$l_{m,j}$	Number of mobile time slots needed to process job (m, j)
$r_{m,j}$	Number of cloud server time slots needed to process job (m, j)
$u_{m,j}$	Number of time slots needed to upload the cloud executed data of job (m, j)
$d_{m,j}$	Number of time slots needed to download the cloud executed result of job (m, j)
Q_m^l	Energy per time slot needed by user class m for local processing
Q_m^t	Energy per time slot used by user class m during mobile transmission
Q_m^r	Energy per time slot needed by user class m during mobile reception
$x_{m,j}$	Binary variable determining if job (m, j) is offloaded for cloud execution
$y_{m,j,t}$	Binary variable determining if cloud server time slot t is assigned to job (m, j)
$e_{m,j}$	Energy consumption associated with processing job (m, j)
$t_{m,j}$	Job generation time of job (m, j)
$D_{m,j}$	Hard deadline constraint for job (m, j)

$a_{m,j}$	Arrival time of job (m, j) at the cloud server
$c_{m,j}$	Time at which the cloud server must complete processing job (m, j)
$\mathcal{T}_{m,j}$	Set of time slots at the cloud server that may be used to execute job (m, j)
γ_m	Control variable in making offload decision
$\mathcal{E}_{T,m}$	Running summation of energy consumption of user class m
$\mathcal{E}_{s,m}$	Weighted average energy consumption of user class m
ε	Probability that the completion time of a job exceeds the required deadline
ρ_m	Probability that jobs from user class m are uploaded to the cloud server
E_m	Energy consumption of user class m associated with probability ρ_m

Abbreviations

EDF	Earliest Deadline First
FCFS	First-Come-First-Served
FGFS	First-Generated-First-Scheduled
ILP	Integer Linear Program
MCC	Mobile Cloud Computing
MCO	Mobile Computation Offloading
OLJS	Open Loop Job Submission
γ-EDF	γ-Ratio Accepted Earliest Deadline First
γ-FGFS	γ-Ratio Accepted First-Generated-First-Scheduled

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Mobile Cloud Computing

1.1.1 Overview

Mobile cloud computing (MCC) is rapidly evolving and is already starting to revolutionize mobile user applications. According to a recent study by Cisco Inc (Cisco, 2015), mobile cloud traffic is expected to grow 11-fold over the next five years with a compound annual growth rate of 60 percent. Global cloud applications are projected to account for 90 percent of total mobile data traffic by 2019.

From the cloud point of view, mobile traffic can be categorized into two types: mobile non-cloud traffic and mobile cloud traffic. Typically, mobile non-cloud traffic includes voice calls, text messaging, etc. On the other hand, mobile cloud traffic covers online gaming, social networking, online storage, audio streaming, video streaming and so on. Internet video applications are classified as cloud traffic in the sense that service providers like YouTube and Netflix have already started to use cloud

applications for video delivery, which in turn significantly boosts the growth of mobile cloud traffic and the prosperity of cloud applications.

MCC can be defined as an infrastructure where data storage and data processing take place in cloud-based servers instead of inside mobile devices (Dinh *et al.*, 2013), and thereafter enable the execution of mobile applications on various mobile devices with a rich user experience (Abolfazli *et al.*, 2014). It is often introduced as an integration of mobile users into conventional cloud computing. Compared to the cases where mobile users suffer from resource-constrained problems such as short battery life, limited data processing capability and data storage, the cloud always has abundant resources which can be used to enrich mobile user operation. To better exploit mobile cloud features, for example, MCC can move mobile data processing and storage into the cloud, which may significantly enhance the mobile system performance. Apple's iCloud[1] and Microsoft's Azure[2] are commercial examples that currently exploit MCC.

MCC has two major functions. One of them is storage backup, which allows mobile devices such as smartphones, tablets and laptops to store data, files, documents, messages and the like in the cloud-based storage. For instance, Dropbox[3] is one of most popular cloud storage service providers which is widely used by not only individuals but also companies. At the same time, a number of such service providers also offer mobile apps along with their products that synchronize all kinds of data across multiple platforms so that users are able to obtain access to the cloud storage from different places.

The other function of MCC is computation offloading, which refers to migrating

[1] `http://www.apple.com/icloud/`
[2] `http://azure.microsoft.com/`
[3] `http://www.dropbox.com/`

computation tasks to external clouds that handle the execution of programs. To be more explicit, mobile users make use of the computing cycles provided by the cloud vendors in order to cut down the computation performed on the mobile devices (Kumar and Lu, 2010). As a consequence, computation capability is no longer limited to the mobile devices, and resources such as CPU, memory and battery that mobile devices invest in processing computation tasks can be saved. A case in point is Amazon Elastic Compute Cloud[4] which provides reliable and secured cloud computing capacity to developers.

Even though MCC is built on the basis of standard cloud computing models including Infrastructure as a Service (IaaS), Platform as a Service (PaaS) as well as Software as a Service (SaaS) (Zhang *et al.*, 2010), it is different from standard cloud computing in various aspects. Most importantly, compared to standard cloud computing by which information technology services and resources are expended in a cost efficient fashion (Khan *et al.*, 2014), MCC emphasizes issues like wireless network connectivity, bandwidth utilization cost and mobile device energy, which are all out of the main scope of standard cloud computing. Nevertheless, they have concerns in common as well, such as security, privacy and stability problems.

1.1.2 Architectures

As a combination of inter-disciplinary approaches comprising mobile computing, cloud computing and mobile networks, MCC integrates these three parts into an entirety. Specifically, there are three types of MCC architecture that specify how mobile devices and the cloud can be connected.

[4] `http://aws.amazon.com/ec2/`

Centralized cloud

In this case the cloud resource is placed in a remote centralized cloud infrastructure, from which mobile devices obtain cloud services via cellular or Wi-Fi networks. The cloud under this circumstance works as an intermediary between content providers and mobile devices (Liu *et al.*, 2013).

Cloudlet

Cloudlet is a mobility-enhanced small-scale cloud data center that can be located in cellular base stations or Wi-Fi access points in order to provide cloud service with lower latency than remote centralized clouds. This idea first appears in (Satyanarayanan *et al.*, 2009).

Ad hoc mobile cloud

Ad hoc mobile cloud pools the nearby mobile devices together and processes computation tasks in a distributed and collaborative fashion. One example of virtual cloud computing provider has been given in (Huerta-Canepa and Lee, 2010) which demonstrates the usefulness of this kind of architecture.

1.1.3 Application Models

There are a number of goals that MCC strives to achieve for the performance improvement of mobile devices, which include providing mobile devices with more powerful execution capability for complicated tasks, decreasing the energy consumption of local execution at mobile devices for longer battery life, reducing the latency required to deliver services to mobile devices, and so on. In some cases,

multiple objectives can be achieved simultaneously. In other cases, however, part of the effectiveness has to be sacrificed for the sake of a given primary goal. So far a wide range of application models have been investigated that are designed to satisfy different objectives.

MAUI

The primary concern of MAUI (Cuervo *et al.*, 2010) is to reduce the energy consumption of mobile devices by offloading code execution to the nearby infrastructure or cloud, based on the assumption that the overhead of code offloading is less than that of executing the same program locally at mobile devices. MAUI also takes the advantage of dynamic partitioning that supports fine-grained code offloading schemes for applications without much programmer intervention.

Clonecloud

Clonecloud (Chun *et al.*, 2011) also aims to minimize the mobile energy consumption based on dynamic application partitioning, where a graph of static flow control is plotted to describe the way of partitioning. Specifically, five types of augmented execution (Chun and Maniatis, 2009) are supported by Clonecloud, which consist of primary functionality outsourcing, background augmentation, mainline augmentation, hardware augmentation and augmentation through multiplicity.

ThinkAir

In terms of executing multiple smartphone clones in parallel, ThinkAir (Kosta *et al.*, 2012) focuses on minimizing energy consumption as well as reducing execution delays. By referring to the history-based information such as energy consumption and execution locations, a controller is used to determine whether a task is executed by the mobile device or at the cloud.

μCloud

The most notable strength of μCloud (March *et al.*, 2011) is that it supports flexibility, reusability and configurability for heterogeneous components of applications. To tackle the problem, μCloud first mimics the applications using directed graphs, and the output of a component is then put into the subsequent components following the graph flow. In this sense, application components can be decoupled from each other, which makes task execution easier.

Cuckoo

Cuckoo (Kemp *et al.*, 2012) is the first practical implementation for Android platform, which supports partial application offloading to the cloud. It builds up a dynamic runtime system in order to make decisions on either local execution or cloud execution for mobile applications. In addition, it simplifies the process of developing smartphone applications thanks to computation offloading.

For a clear comparison, Table 1.1 shows how five of the above mentioned application models can be effectively used in dealing with different scenarios (Khan

Table 1.1: MCC application examples

Application Model	MAUI	Clonecloud	ThinkAir	μCloud	Cuckoo
Mathematical Calculation	√		√	√	√
File Search		√			
Image Processing	√		√	√	√
Gaming	√		√	√	√
Anti-virus		√			

et al., 2014).

In practice, one of the representative mobile cloud platforms is Qualcomm's Vuforia Cloud Recognition[5], which is a mobile vision platform that provides recognition service to supercharge mobile apps with a large number of targets. It is able to not only expand the mobile recognition and tracking capabilities in objects including images and text, but also perform vision-based computing such as video playback. A lot of relevant mobile apps have also been developed to interact with various surfaces and objects.

1.2 Energy Saving for Mobile Devices

With the evolution of mobile communication systems, mobile users expect better service quality and user experience. However, mobile devices are inherently resource-poor, which limits the capability of providing enhanced services. To be more explicit, the physical size of mobile devices sets strict constraints on the supply of resources in terms of battery, computation and storage.

Among all of the weaknesses, short battery life has become the most challenging issue (Miettinen and Nurminen, 2010) and the biggest complaint (Kumar and Lu,

[5]http://www.qualcomm.com/products/vuforia

2010) for mobile devices. Nevertheless, more computation-intensive applications, such as gaming, navigation, voice recognition and image processing, are being developed and installed on mobile devices, which exploit and exhaust the battery at a much higher rate. Since battery development is encountering a bottleneck period, energy saving strategies (Vallina-Rodriguez and Crowcroft, 2013) are instead taken into consideration for the design of emerging mobile systems, in order to release the stress between resource-poor mobile devices and resource-intensive applications.

One of the feasible solutions is to migrate computation tasks from mobile devices to other machines like cloud servers, so that computation is handled outside the mobile devices and the devices' battery lifetime can therefore be extended. Towards this end, computation offloading is proposed and implemented in the framework of MCC systems. For example, (Kumar and Lu, 2010) gives a couple of representative examples demonstrating how energy consumption of mobile devices can be saved with the help of computation offloading, where calculation offloading in chess games is considered for energy saving. Also, cloud-based anti-virus applications like CloudAV (Oberheide *et al.*, 2008) are expected to consume a minimal amount of energy in mobile systems when adopting the optimal execution strategy proposed in (Zhang *et al.*, 2013). Such kinds of practical applications shed light upon the full implementation of computation offloading in MCC, which works towards the ultimate goal of energy reduction.

However, computation offloading might not be an effective solution of energy saving in every case, because its performance is also dependent on different types of applications. Due to the various degrees of local computation and offload communication in different mobile applications, computation offloading is preferable

only for a certain part of them, and local execution should be considered for the rest. In practice, other than the well-known example of chess games that highlights the benefit of computation offloading in dealing with computation-intensive applications, there are also lots of applications that are energy-inefficient if using cloud execution. For instance, (Namboodiri and Ghose, 2012) shows that playing audio and video files in formats of MP3, FLV and MPEG2 using cloud incurs a greater amount of energy than using local execution, which demonstrates the limited use of MCC. Moreover, (Kumar and Lu, 2010) investigates three critical factors in network settings that play an important role of determining offloading decisions, which include the wireless bandwidth, the amount of computation and the amount of transmission data. As a result, high offload communication and small bandwidth turn out to decrease the attractiveness of computation offloading. Therefore, it is of great significance to make a reasonable choice between computation offloading and local execution such that mobile devices can achieve a desirable level of energy consumption.

1.3 Challenges

MCC is a promising technology of providing mobile users with enhanced performance. However, there are also several challenging issues which need to be addressed in order to make the system function in a satisfactory manner.

1.3.1 Latency

Typically, latency in MCC is referred to as the time spent from job uploading to cloud-executed result downloading. The latency may vary significantly as a result of different task size, partitioning scheme, and execution delay.

More importantly, wireless channel conditions also play a nontrivial role of determining the latency because severe channel fading can significantly lower the wireless transmission rate and correspondingly raise the latency. In particular, latency is a crucial criteria for real-time applications like online gaming and video streaming of which the user experience is highly sensitive.

Firewalls and overlay networks inevitably increase the software path length that offloaded data has to go through (Satyanarayanan *et al.*, 2009), which makes it more difficult to obtain shorter latency. In addition, employing energy-saving techniques like intermittently turning on the mobile device's transceiver also increases average end-to-end latency, which further indicates the disagreement between energy saving and latency reduction.

1.3.2 Bandwidth

In the cases where data is uploaded to the cloud, or files are downloaded from the cloud, large bandwidths may be required. However, bandwidth is far more costly in wireless networks than in wired networks.

In the meantime, latency is largely dependent on bandwidth as well. High bandwidth usage can benefit the latency performance in terms of short offload time. Therefore, service providers need to work carefully towards an eclectic interaction among bandwidth utilization, costs and latency.

1.3.3 Reliability

Reliability refers to reliable connectivity to the cloud server. Due to the fact that MCC relies heavily on wireless communication networks including cellular and Wi-Fi, it is of vital importance to make sure that reliable connections can be provided, particularly when data is not stored locally.

On the other hand, seamless handoff between heterogeneous networks should also be addressed. Since a mobile device is allowed access to cellular base stations, Wi-Fi access points, femtocells and so on, continuous transmission connectivity between the mobile device and the cloud is required regardless of the network being used. Additionally, mobile users have the right to choose the network they intend to use when multiple networks are available.

1.3.4 Security

Bugs and loopholes at the cloud are two of the most dangerous threats to the security of MCC, especially because they are usually hard to discover. The data stored in the cloud might also be denied or lost because of malfunctions. Consequently, MCC service providers need to cautiously treat the security concerns related to the data that mobile users upload.

Mobile users should also make sure that their mobile devices are protected from viruses and malwares. Installing anti-virus applications and software is also suggested whereas scanning the mobile device is likely to incur high energy consumption. Even though anti-virus scan can currently be performed at the cloud, it alternatively requires to pre-synchronize all the files.

1.3.5 Privacy

Privacy is a hot topic which has attracted more attention in the recent decade. As an increasing number of files, documents and messages are now being stored in the cloud, the cloud can never afford privacy leakage, which raises critical requirements for privacy protection. Although methods such as encryption and steganography are available to encipher private information, performing complicated calculation algorithms can consume a lot of energy on mobile devices, which offsets the advantage of MCC.

Privacy concerns also exist on the side of mobile devices. One of the examples is the GPS locator which takes records of the geographical locations of the mobile user that are possibly abused by applications installed on the mobile device. Moreover, different regulations and standards among countries might also lead to discrepancies in the level of privacy protection.

1.3.6 Platform

Platform is the fundamental software technology on which mobile applications are based. Some popular mobile operating systems include Android, iOS and Windows Phone. However, each of these operating systems has its distinctive features. For instance, Android supports applications in Java, while iOS accepts Objective C exclusively. Therefore, MCC applications need to be designed in different ways in order to be adapted to various platforms. The level of support for computational offloading has also triggered the heterogeneity of applications, with a representative fact being that iOS is less supportive than Android (Khan *et al.*, 2014).

1.4 Literature Review

1.4.1 MCC Systems and Applications

There is a wide variety of recent works on MCC. Reference (Guan *et al.*, 2011), (Fernando *et al.*, 2013) and (Dinh *et al.*, 2013) conduct comprehensive surveys and work out specifications about MCC systems. Reference (Liu *et al.*, 2013) presents three major MCC architectures including centralized cloud, cloudlet and ad-hoc cloud configurations. At the same time, (Magurawalage *et al.*, 2014) proposes a mechanism for choosing between centralized clouds or cloudlets, and (Huerta-Canepa and Lee, 2010) discusses the mobility of ad-hoc mobile cloud based infrastructure. These studies show that MCC can be beneficial to mobile users in various ways. Reference (Xu and Wen, 2013) discusses how MCC performs over Wi-Fi and 3G cellular, and (Lei *et al.*, 2013) investigates practical issues involving the support of MCC in heterogeneous networks. Additionally, (Satyanarayanan *et al.*, 2009) reviews a range of future challenges of MCC.

In order to obtain a better service from mobile clouds, lots of issues related to MCC are also tackled. Reference (Fan *et al.*, 2011) makes a thorough discussion on mobile cloud application models, while different cloud-based application models are compared in detail in (Kovachev *et al.*, 2011). Some other works such as (Liang *et al.*, 2012) view MCC from an economical standpoint towards a goal of offering affordable cloud service to mobile users.

In recent years, a great number of applications based on the notion of MCC are emerging as well, which provide cloud-based services in a variety of fields, ranging from education (Zhao *et al.*, 2010) (Ferzli and Khalife, 2011) to commerce (Yang

et al., 2010) and from searching (Pendyala and Holliday, 2010) to gaming (Cuervo *et al.*, 2010). MCC based applications also provide effective solutions towards new topics appearing in the new century, such as mobile social TV systems (Wu *et al.*, 2013) and healthcare systems (Tang *et al.*, 2010) (Doukas *et al.*, 2010). Moreover, (Wan *et al.*, 2013) integrates wireless body area network with cloud based mobile systems for healthcare applications.

1.4.2 Computation Offloading

The objective of computation offloading is to migrate computations from resource-limited mobile users to resourceful cloud servers, thereby avoiding the energy use that is required for local execution (Elgazzar *et al.*, 2015). Reference (Zhu *et al.*, 2013) demonstrates the feasibility of computation offloading in MCC, and the studies in (Rudenko *et al.*, 1998) and (Khan *et al.*, 2014) show that remote application execution can significantly reduce mobile energy use in this type of situation. Reference (Kumar and Lu, 2010) gives a series of mathematical formulations that show how computation offloading can reduce energy use in a mobile device, whilst experiments given in (Rudenko *et al.*, 1998) and (Rudenko *et al.*, 1999) show that remote processing is able to save as much as half of the battery life with a few realistic applications, which further indicates the prospect of computation offloading in energy saving.

The benefits of computation offloading in extending battery life are studied in (Othman and Hailes, 1998) and (Rong and Pedram, 2003) in early years, in which remote processing at that time is akin to computation offloading in present MCC frameworks. Some recent works pinpoint the effectiveness of computation offloading on energy saving for mobile battery, such as (Huang *et al.*, 2012) and (Xian *et al.*,

2007) which propose dynamic and adaptive approaches that determine the ways in which computation offloading is performed.

Depending on the type of applications and the feasibility of partitions, mobile computation jobs can be offloaded to the cloud server either entirely or partially. Reference (Zhang *et al.*, 2013) proposes a method for optimally executing mobile applications either locally or in the cloud assuming varying wireless channel conditions. In contrast to executing the entire job either locally or using a cloud server, job partitioning, where different local and cloud job partitions can be selected is considered in (Gu *et al.*, 2004), (Huang *et al.*, 2012) and (Ma *et al.*, 2013). Several papers including (Cuervo *et al.*, 2010), (Chun *et al.*, 2011) and (Kosta *et al.*, 2012) consider energy efficiency when cloud execution and local execution partitions are selected.

1.4.3 Job Scheduling

When multiple jobs from different mobile users arrive at the cloud, they need to be queued at the cloud server before being processed. At the same time, the cloud server must devise scheduling algorithms that work for certain goals such as accepting a maximized number of jobs or decreasing the average queueing time across all jobs. However, complicated scheduling algorithms require a considerable amount of calculation and memory space, which inevitably increases the overhead of the server. Therefore, queueing and scheduling issues should be carefully managed for the sake of system performance.

Job scheduling plays an important role of achieving satisfactory system performance, particularly in the cases where multiple users submit jobs to the same

server and correspondingly compete for the server resources. A wide range of models related to queueing and scheduling strategies are investigated, where the queue length and waiting time can be predicted mathematically. Some examples of job scheduling disciplines include first-come-first-served (FCFS) (Schwiegeishohn and Yahyapour, 1998), processor sharing (Parekh and Gallager, 1993), and earliest deadline first (EDF) (Kruk *et al.*, 2004).

Further, there are some works that address the activity of multiple users competing for cloud server resources, and examples of these include (Barbarossa *et al.*, 2013), (Kaewpuang *et al.*, 2013) and (Chen, 2015). Reference (Huang *et al.*, 2014) specifically focuses on job scheduling in a resource intensity aware scenario. Also, an optimization problem of radio and computational resources in multi-cell situations is investigated in (Sardellitti *et al.*, 2014), while (Song *et al.*, 2014) and (Achary *et al.*, 2015) achieve improvements in task scheduling using Lyapunov optimization and ant colony optimization. In addition, (Maguluri and Srikant, 2013) takes unknown job size into consideration and accordingly works out a set of throughput-optimal job scheduling solutions, and (Lin *et al.*, 2014) further presents the solution of minimal-delay task scheduling.

1.5 Overview and Organization of the Thesis

In this thesis, we consider mobile users that access an infrastructure-based cloud server, and use computation offloading to reduce their energy consumption. Arriving jobs are assumed to have hard deadline completion constraints. In a wireless network, the job completion times include the latencies needed for uploading and downloading the cloud-executed job components and results,

respectively. This overhead can introduce significant energy unfairness, where for example, a mobile user with poor channel conditions may be prevented from engaging in computation offloading due to the added latency. This energy unfairness can be compensated for, using smart scheduling at the cloud server, by offering differentiated service to those users that are in energy disadvantaged situations. The design of these types of schedulers is the main focus of the thesis.

An offline scheduler is first derived using an integer linear program (ILP) formulation that makes optimum job submission selections. This formulation uses a min-max energy objective with preemptive cloud server scheduling. The computational complexity of optimal min-max computation offloading is then shown. We then introduce several online scheduling algorithms. The first is referred to as First-Generated-First-Scheduled (FGFS), which performs job scheduling on the basis of job generation time. A modified version, referred to as γ-Ratio Accepted FGFS (γ-FGFS), is then proposed where the acceptance of a job submission is subject to an energy threshold test. We also consider versions of the schedulers that include Earliest Deadline First (EDF) scheduling. EDF inherently helps to improve computation offloading energy unfairness due to differing channel conditions. These variants are referred to as γ-Ratio Accepted EDF (γ-EDF) scheduling. The thesis presents a wide variety of scheduler comparisons, which show the value of smart scheduling from an energy fairness viewpoint. We also include comparisons using an analytic model that shows the performance possible when open loop job submission (OLJS) is used with first-come-first-served (FCFS) cloud server scheduling. In this model, mobile job submission feedback is not provided, and therefore the user job submission rate is flow controlled to achieve a desired level of performance.

The rest of the thesis is organized as follows. In Chapter 2 we give a detailed description of the system model and formulate the optimum offline scheduler. Computational complexity of the optimum scheduling is also analyzed. Online scheduling algorithms are then designed in Chapter 3, followed by an open-loop scheduler proposed in Chapter 4 for comparison. Performance results for all the algorithms are given in Chapter 5. Finally, Chapter 6 contains the conclusions of our work and possible future research topics.

Chapter 2

System Model and Problem Formulation

2.1 Overview

In this chapter we first describe the MCC system that our research is based on. We then formulate a problem whose solution provides the optimum scheduling in an offline fashion and serves as a lower bound of energy consumption on the performance of online scheduling algorithms. We also show that the optimum offline scheduling problem is NP-complete.

2.2 System Model

The system considered is shown in Figure 2.1, where M classes of mobile users request job execution services from an infrastructure-based cloud server. The mobile users within each class transmit and receive using the same wireless transmission data

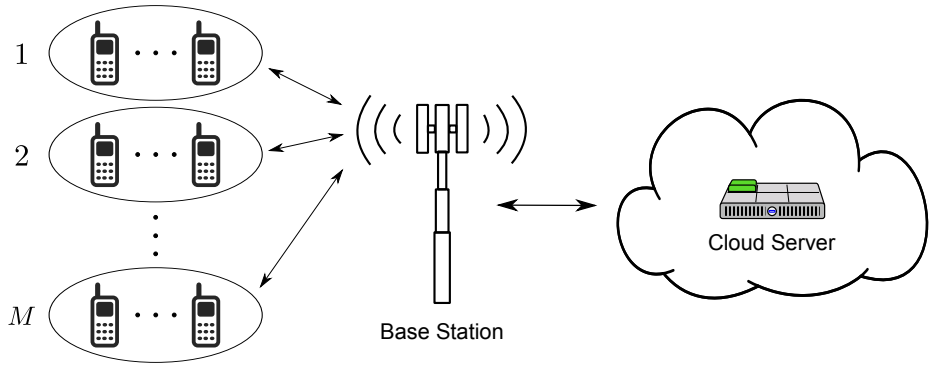

Figure 2.1: Mobile cloud computation offloading model. M classes of mobile users access cloud server execution services and use computation offloading.

rate. The set of user classes is denoted by $\mathcal{M}$ and indexed by $m \in \{1, 2, \ldots, M\}$. The system is modeled in discrete time, where each time interval is referred to as a time slot, and indexed by $t \in \{1, 2, \ldots, T\}$. The complete set of time slots considered is defined by $\mathcal{T}$, and over this time, mobile users in class m generate J_m jobs indexed by $j \in \{1, 2, \ldots, J_m\}$. The set of jobs generated by mobile users in class m is denoted by $\mathcal{J}_m$, where the jth job is referred to as job (m, j).

Each job can be executed either locally at the mobile user, or uploaded to, and executed by the cloud server as in (Gu *et al.*, 2004), i.e., we consider the binary partition case. We define $l_{m,j}$ and $r_{m,j}$ as the numbers of mobile and cloud server time slots needed to process job (m, j), respectively. If the job is uploaded, $u_{m,j}$ and $d_{m,j}$ give the corresponding numbers of time slots needed to upload and download the cloud executed data and results, respectively.

We are interested in the mobile energy consumption when an offloading decision is considered. The energy consumed for uplink transmission of job (m, j) is defined by $Q_m^t u_{m,j}$, where Q_m^t is the energy per time slot used during transmission for mobile class m. When the cloud server returns results, the mobile energy required is defined

by $Q_m^r d_{m,j}$, where Q_m^r is the energy per time slot needed during mobile reception. Similarly, the energy associated with the local processing of job (m, j) is given by $Q_m^l l_{m,j}$, where Q_m^l is the per time slot energy needed for local processing at a mobile user of class m. The energy consumption associated with processing job (m, j) is therefore given by

$$e_{m,j} = (1 - x_{m,j})Q_m^l l_{m,j} + x_{m,j}\left(Q_m^t u_{m,j} + Q_m^r d_{m,j}\right) \tag{2.1}$$

for all $m \in \mathcal{M}$ and $j \in \mathcal{J}_m$, where $x_{m,j}$ is a set of binary decision variables defined as

$$x_{m,j} = \begin{cases} 1 & \text{if job } (m,j) \text{ is offloaded for cloud execution,} \\ 0 & \text{otherwise.} \end{cases}$$

For a job (m, j), we denote its generation time as $t_{m,j}$. Its completion is subject to a hard deadline constraint, given by $D_{m,j} \in \mathcal{T}$. If the job is offloaded, its arrival time at the cloud server is given by

$$a_{m,j} = t_{m,j} + u_{m,j}, \tag{2.2}$$

and the time at which the cloud server must complete processing the job is then given by

$$c_{m,j} = D_{m,j} - d_{m,j}. \tag{2.3}$$

Here, it is assumed that at each user, a job is processed before a new job is generated. We define $\mathcal{T}_{m,j} \triangleq [a_{m,j}, c_{m,j}]$ to be the range of time slots at the cloud server that may be used to execute job (m, j), i.e., this job must be executed using $r_{m,j}$ time slots

from the set $\mathcal{T}_{m,j}$.

In addition to deciding whether a given upload request can be accepted, i.e., the value of $x_{m,j}$, a job scheduler must also decide how the accepted job will be processed by the cloud server, so that the job deadline constraints are satisfied. The cloud server execution schedule is obtained by assigning values to the following set of binary decision variables

$$y_{m,j,t} = \begin{cases} 1 & \text{if cloud server time slot } t \text{ is assigned to job } (m,j), \\ 0 & \text{otherwise.} \end{cases}$$

The scheduling objective is to assign values for $x_{m,j}$ and $y_{m,j,t}$ to achieve the following min-max energy usage

$$\min \max_{m} \sum_{j \in \mathcal{J}_m} e_{m,j}.$$

This objective will achieve energy fairness among the mobile user classes that compete for use of the cloud server. The scheduling will try to favour those classes that are experiencing unfavourable energy conditions.

Noticeably, mobile computation offloading (MCO) can be performed in either a closed or open-loop manner. In closed-loop MCO, the mobile users interact with the cloud server in real-time as jobs arrive. When a job is generated, the mobile makes an offload request to the cloud server, including the remote execution job size and processing deadline. A feasibility check is performed at the cloud server, which informs the mobile of the acceptance decision. If the offloading request is accepted, its execution is also scheduled at the cloud server, and job uploading begins. Otherwise, the job is executed locally.

In contrast, open-loop MCO does not involve a real-time negotiation with the cloud server. Instead, the users must flow control their job submissions in order to achieve a desired level of performance. Because of the lack of cloud server interaction, it is generally not possible to achieve hard deadline constraints with open-loop MCO, and for this reason, closed-loop MCO is the main focus of the thesis, and several closed-loop scheduling algorithms are proposed in Chapter 3. However, for comparison purposes, an analytic model is formulated in Chapter 4 that can be used to determine the optimum min-max energy performance of an open-loop MCO system, which is subject to statistical delay constraints.

2.3 Optimum Offline Scheduling

2.3.1 Offline Scheduling Formulation

In offline scheduling, the complete set of inputs over all time is provided to the scheduler all at once. Based on the definitions in Section 2.2, we formulate an integer linear program (ILP) that gives an optimal offline min-max energy schedule, assuming preemptive scheduling. The ILP, referred to as (OPT), is first given, and then discussed below.

$$\underset{x_{m,j},y_{m,j,t}}{\text{minimize}} \quad \sum_{m\in\mathcal{M}} \sum_{j\in\mathcal{J}_m} e_{m,j} \tag{OPT}$$

subject to

$$\sum_{j\in\mathcal{J}_m} e_{m,j} \leq \widehat{\mathcal{E}}, \quad \forall m \in \mathcal{M} \tag{2.4}$$

$$\sum_{t\in\mathcal{T}_{m,j}} y_{m,j,t} \geq x_{m,j}\, r_{m,j}, \quad \forall m \in \mathcal{M}, j \in \mathcal{J}_m \tag{2.5}$$

$$\sum_{m \in \mathcal{M}} \sum_{j \in \mathcal{J}_m} y_{m,j,t} \leq 1, \quad \forall t \in \mathcal{T} \tag{2.6}$$

$$x_{m,j} \in \{0,1\}, \quad \forall m \in \mathcal{M}, j \in \mathcal{J}_m \tag{2.7}$$

$$y_{m,j,t} \in \{0,1\}, \quad \forall m \in \mathcal{M}, j \in \mathcal{J}_m, t \in \mathcal{T} \tag{2.8}$$

The objective function in (OPT) computes the minimum total energy over the given set of inputs, where $e_{m,j}$ is given in (2.1). Constraint (2.4) however, places a common upper bound, $\widehat{\mathcal{E}}$, on the total energy used by each mobile class. Since $\widehat{\mathcal{E}}$ is not known a priori, we can do a binary search on its value, solving (OPT) each time, to get the minimum value of $\widehat{\mathcal{E}}$ that achieves an optimal min-max energy bound, i.e., a bound on the best mobile energy loading possible. This formulation therefore finds the best min-max solution while minimizing total energy. Constraint (2.5) ensures that the number of time slots assigned when job (m, j) is uploaded (i.e., $x_{m,j} = 1$) is at least $r_{m,j}$. Note that when this job is executed locally (i.e., $x_{m,j} = 0$), this constraint is satisfied by any set of $y_{m,j,t}$ assignments. Constraint (2.6) restricts the schedule so that at most one job can be assigned to a given cloud time slot. Finally, (2.7) and (2.8) define the binary decision variables.

Optimization (OPT) can be altered in various ways depending on the desired min-max energy criterion. For example, in some cases, it may be desirable to normalize the energy expended to the job size, which may be accomplished by dividing each term in (2.4) by the number of job units associated with job (m, j). This example would focus on energy fairness that does not unduly prefer users with higher loading than the others. On the other hand, the formulation above is preemptive. That is, before finishing the execution of a given job, the server can switch to serve another job and return to process the first job later on. Alternatively, the server can be restricted

to serve jobs without preemption, and this has been studied in (Yue *et al.*, 2014b) where we have formulated an optimization problem that gives an optimum offline scheduling solution and proposed online scheduling algorithms.

2.3.2 Complexity Analysis

The formulation above gives the optimum min-max energy computation offloading for an offline scheduler, and is therefore a lower bound on the performance of any online scheduler. In this part we prove the following theorem that confirms the exponential worse-case complexity of the min-max energy computation offloading problem.

Theorem 1. *Optimum min-max energy computation offloading with job deadlines is an NP-complete problem.*

Proof. The proof is by a reduction to the PARTITION problem. In PARTITION, we are given a set of N integers, denoted by $\mathcal{S} = \{x_1, x_2, \ldots, x_N\}$ and where $X = \sum_{x \in \mathcal{S}} x$. To solve PARTITION, we must answer the following question: *Is there a set, $\mathcal{S}' \subset \mathcal{S}$, such that $\sum_{x \in \mathcal{S}'} x = X/2$?*

To create the reduction, assume that we are given an instance of PARTITION, as described above. We consider a system with a single mobile user class, i.e., user class 1, that generates a set of $J_1 = N$ jobs to be processed starting at $t = 0$. We assign $u_{1,j} = x_j$ and $r_{1,j} = 0$ for all $j \in J_1$. Under these circumstances, once a job is finished uploading, the remote cloud execution time is zero, i.e., the job execution is completed. All job deadlines are set to $D_{1,j} = X/2$ for all $j \in J_1$. We also assume that the local execution components for each job are such that all N jobs can be executed locally within the specified deadline. Since Q_1^l is an independent variable, it can be set sufficiently large so that the more job components that are uploaded

rather than executed locally, result in a reduction in the total energy required. Note that with these parameter settings, the total energy needed for execution is a strictly decreasing function of the uplink channel utilization.

Now assume that we have an optimal min-max computation offloading algorithm. We use it to solve the system described above using the PARTITION instance inputs. If an equal partition of S exists, our algorithm must find it. Otherwise, the total upload time will be less than $X/2$, and therefore, the total energy will not be minimized, i.e., an upload schedule that partitions S equally would achieve lower energy, which would contradict the assumed optimality of our algorithm. Therefore, if the upload time from our optimization is $X/2$, then a partition of S exists. Otherwise, a partition of S does not exist. □

2.4 Summary

In this chapter we described the system model and formulated the problem of optimum offline scheduling. This scheduling however, is not causal, as it requires future information at the time of making scheduling decisions. In order to address this problem, heuristic scheduling that can be implemented in real systems should be designed. Several online scheduling algorithms based on closed-loop MCO are proposed in Chapter 3. We also introduce an online scheduler associated with open-loop MCO in Chapter 4 for comparison.

Chapter 3

Online Scheduling

3.1 Overview

In online scheduling, the inputs are provided to the scheduler in real time and the scheduler must assign the values of $x_{m,j}$ and $y_{m,j,t}$ in a causal fashion, based solely on past and current inputs.

In this chapter, several online cloud server scheduling algorithms are introduced, where mobile users interact with the cloud server in real-time. As in offline scheduling, the objective of online scheduling is to define algorithms that can achieve energy fairness among mobile user classes, subject to job completion deadline constraints. In addition, the complexity property of all online algorithms is analyzed.

3.2 First-Generated-First-Scheduled (FGFS)

In FGFS scheduling, jobs that are generated earlier are scheduled first at the cloud server. Unlike many conventional scheduling models, in our case, the job arrival at

Algorithm 1 FGFS

1: A job (m, j) is generated at time $t_{m,j}$. Mobile m sends an upload request to the cloud server.
2: The cloud server checks the following condition

$$\sum_{t \in \mathcal{T}_{m,j}} \sum_{m' \in \mathcal{M}} \sum_{j' \in \mathcal{J}_{m'}} (1 - y_{m',j',t}) \geq r_{m,j}.$$

3: **if** the condition is satisfied **then**
4: $\quad x_{m,j} = 1$
5: $\quad$ initialize $\widetilde{n} = 0$ and $t = a_{m,j}$
6: $\quad$ **while** $\widetilde{n} < r_{m,j}$ **do**
7: $\quad\quad$ **if** $\sum_{m' \in \mathcal{M}} \sum_{j' \in \mathcal{J}_{m'}} y_{m',j',t} = 0$ **then**
8: $\quad\quad\quad y_{m,j,t} = 1$
9: $\quad\quad\quad \widetilde{n} = \widetilde{n} + 1$
10: $\quad\quad$ **end if**
11: $\quad\quad t = t + 1$
12: $\quad$ **end while**
13: **else**
14: $\quad x_{m,j} = 0$
15: **end if**
16: **return**

the cloud server occurs *after* the job upload has completed. For this reason, in FGFS, the job is scheduled when it is generated by the user, before the upload occurs. For a given job arrival, the upload decision is determined by a feasibility test based on the job deadline. The procedures are defined in Algorithm 1 and described below.

When a job (m, j) is generated, the user sends an upload request as shown in line 1, which includes parameters $D_{m,j}$, $r_{m,j}$ and $d_{m,j}$. Such a step might also incur a certain level of mobile energy consumption in sending the upload request and receiving the feasibility test result. However, it is fairly small compared with the energy usage of processing the job locally or offloading this job to the cloud.

In line 2, the feasibility test checks if there are at least $r_{m,j}$ available time slots

within the set $\mathcal{T}_{m,j}$, i.e., if the deadline of job (m, j) can be satisfied based on the current time allocation at the cloud server. If the feasibility test is satisfied, the job can be uploaded; otherwise the mobile user processes the job locally. Lines 5-12 give details of allocating the earliest $r_{m,j}$ free time slots for job (m, j) within the time interval $\mathcal{T}_{m,j}$, and $x_{m,j} = 1$ is accordingly assigned.

The complexity of the FGFS scheduler can be easily found. When job (m, j) arrives, FGFS must find $r_{m,j}$ free time slots from the set $\mathcal{T}_{m,j}$. The complexity of FGFS is therefore linear in the cardinality of this set, i.e., $\mathcal{O}(|\mathcal{T}_{m,j}|)$.

3.3 γ–FGFS

In the FGFS scheduler, the uploading request from a user is accepted provided that the deadline constraint can be met. This process does not explicitly differentiate the users on the basis of their past energy performance. A drawback of this is that previously accepted jobs may unnecessarily occupy the cloud server, resulting in higher energy consumption for subsequent jobs, especially from users who are energy disadvantaged. In γ-FGFS scheduling, the cloud server keeps a running estimate of the energy consumption of each user class m, $\mathcal{E}_{T,m}$ and $\mathcal{E}_{s,m}$, and this information is used in making the offload decisions, where $\mathcal{E}_{T,m}$ is the total energy consumption of class m users up to the current time, and $\mathcal{E}_{s,m}$ is a weighted average energy of class m up to the current time.

For a given job (m, j), the mobile user calculates its energy consumption $e_{m,j}$ using (2.1), assuming the job is executed locally. The cloud server also keeps the maximum energy consumption of all the mobile user classes, i.e., $\mathcal{E}_{\max} = \max_m \mathcal{E}_{T,m}$. With these definitions, the procedures of γ-FGFS are given in Algorithm 2, where γ_m

Algorithm 2 γ-FGFS

1: A job (m, j) is generated at time $t_{m,j}$. Mobile m sends an upload request to the cloud server.
2: **if** $\mathcal{E}_{T,m} + e_{m,j}|_{x_{m,j}=0} < \gamma_m \mathcal{E}_{\max}$ **then**
3: $\quad x_{m,j} = 0$
4: **else**
5: $\quad$ Execute lines 2-15 in Algorithm 1
6: **end if**
7: Update the per class energy:
$\quad \mathcal{E}_{T,m} \leftarrow \mathcal{E}_{T,m} + e_{m,j}$
$\quad \mathcal{E}_{s,m} \leftarrow (1-\theta)\mathcal{E}_{s,m} + \theta e_{m,j}$
where $\theta \in (0, 1)$.
8: **if** $\mathcal{E}_{s,m} \geq \overline{\mathcal{E}}_s$ **then**
9: $\quad \gamma_m = \max\{0, \gamma_m - 1/(j+1)^\kappa\}$
10: **else**
11: $\quad \gamma_m = \min\{1, \gamma_m + 1/(j+1)^\kappa\}$ with probability $1 - \mathcal{E}_{s,m}/\overline{\mathcal{E}}_s$, where $\kappa \in (0.5, 1]$.
12: **end if**
13: Update the global maximum energy:
$\quad \mathcal{E}_{\max} \leftarrow \max\{\mathcal{E}_{\max}, \mathcal{E}_{T,m}\}$
$\quad \overline{\mathcal{E}}_s \leftarrow \text{mean}\{\mathcal{E}_{s,m}, \forall m\}$
14: **return**

is a control variable that is dynamically adjusted.

Initially, γ_m is randomly selected between 0 and 1. The procedures start with an upload request from the mobile, as shown in line 1. However, instead of checking the feasibility immediately, in line 2, the cloud server checks if the energy consumption of processing the job locally is less than $\gamma_m \mathcal{E}_{\max}$. If so, the mobile user is instructed to process the job locally; otherwise, the same procedures as lines 2-15 in Algorithm 1 are performed.

In line 7, $e_{m,j}$ is computed, and the values of $\mathcal{E}_{T,m}$ and $\mathcal{E}_{s,m}$ are updated. Here θ is between 0 and 1 to balance the weights of the past and current energy consumption (Yue *et al.*, 2014c). When θ is closer to zero, more weight is given to the past average, so that the result is less affected by the randomness of a single job.

In lines 8 to 12, γ_m is updated. Here $\overline{\mathcal{E}}_s$ is the average value of $\mathcal{E}_{s,m}$ for $\forall m$ and is initialized to 0. If $\mathcal{E}_{s,m}$ is no less than $\overline{\mathcal{E}}_s$, γ_m is decreased by $1/(j+1)^\kappa$; otherwise, γ_m is increased by $1/(j+1)^\kappa$ with a probability $1 - \mathcal{E}_{s,m}/\overline{\mathcal{E}}_s$. Note that the value of γ_m is increased more gradually than during a decrease, since a larger value of γ_m will eventually lead to a higher value of $\mathcal{E}_{\max}$. With the probability given in line 11, a mobile class has a greater chance of increasing γ_m if it is in a good energy situation, as reflected by a larger difference between $\mathcal{E}_{s,m}$ and $\overline{\mathcal{E}}_s$. This updating policy is motivated by, and operates in a manner similar to an evolutionary game (Sahlins and Service, 1988) (Niyato *et al.*, 2009) in the sense that advantaged users are sacrificed in order to compensate for those disadvantaged. Thus, all users work towards a common goal of decreasing the maximum energy use by dynamically adopting a proper strategy in terms of γ_m (Yue *et al.*, 2014a). The step size for adjusting γ_m is $1/(j+1)^\kappa$, which gives a declining exploration probability that is a common choice for approximating the expected value function (Ryzhov *et al.*, 2014). To be more explicit, a small value of κ can speed up the exploration process (Powell, 2007). Finally, in line 13, both of the energy values are updated.

The complexity of γ-FGFS the same as that of FGFS, i.e., $\mathcal{O}(|\mathcal{T}_{m,j}|)$ which occurs in line 5.

3.4 Earliest Deadline First (EDF)

Earliest deadline first is a scheduling algorithm that has been widely used in real-time systems (Jeffay *et al.*, 1991). In (Li, 2006), for example, it is noted that the optimality of EDF is highlighted for periodic and sporadic preemptive tasks. In our case, we adapt EDF for use in scheduling execution offloading. EDF is advantageous

Algorithm 3 EDF

1: A job (m, j) is generated at time $t_{m,j}$. Mobile m sends an upload request to the cloud server.
2: Define $\mathcal{L}$ as the set of *scheduled* jobs with deadlines greater than that of (m, j), and form the set $\mathcal{Q} = \mathcal{L} \cup (m, j)$ sorted in EDF order.
3: $F = 0$
4: **for all** $q \in \mathcal{Q}$ **do**
5: **while** $F = 0$ **do**
6: Execute lines 5-12 in Algorithm 1 for job q
7: **if** $t > c_q$, i.e., a *deadline-feasible* assignment for job q is not obtained **then**
8: $F = 1$
9: **end if**
10: **end while**
11: **end for**
12: **if** $F = 0$ **then**
13: $x_{m,j} = 1$
14: **else**
15: $x_{m,j} = 0$ and scheduling is returned to that before job (m, j) is generated
16: **end if**
17: **return**

in this application since it helps to accommodate users that are energy disadvantaged because of poor channel conditions. Since their jobs take longer to upload, giving them prioritization based on job deadlines helps to compensate for this unfairness. The algorithm description is shown in Algorithm 3 and described as follows.

The scheduler starts as in the FGFS algorithm by sending the uploading request to the cloud server. Before performing the feasibility check and scheduling, a set $\mathcal{Q}$, is formed that contains $\mathcal{L}$, which is the set of jobs that are currently scheduled and awaiting cloud server execution. This set includes the new job request. A binary variable, F, is set to zero before the loop starts. The iteration from lines 4 to 11 loops over the jobs in $\mathcal{Q}$ in EDF order. In a given iteration, the same procedures as lines 5-12 in Algorithm 1 are performed in order to test if a job q can be scheduled so

that its deadline constraint is satisfied. In line 7, if $t > c_q$, job q cannot be scheduled using time slots in $\mathcal{T}_q$, and F is set to 1 so that the iteration does not continue. If the iteration completes and all job deadlines are satisfied, then $F = 0$, and the new request is accepted, i.e., $x_{m,j} = 1$; otherwise, $x_{m,j} = 0$, and the scheduling is returned to that before job (m, j) is generated. This is given in lines 12-16.

In EDF scheduling, $\mathcal{Q}$ is first updated by adding the new job using EDF ordering. Then feasible time slot assignment is done for all jobs in $\mathcal{Q}$. Note, however, that the jobs currently in $\mathcal{Q}$ have already been sorted. In this case, the complexity is given by $\mathcal{O}(|Q| + \sum_{q\in\mathcal{Q}} |\mathcal{T}_q|)$, where $\mathcal{T}_q$ is the set of time slots over which job q must be scheduled. In the homogeneous case, i.e., $|\mathcal{T}_q| = |\mathcal{T}|$ for all q, we have $\mathcal{O}(|\mathcal{Q}|(|\mathcal{T}|+1))$.

3.5 γ-EDF

In order to improve the cloud server job acceptance rate, EDF scheduling can be combined with the γ-based scheduling described previously. This is easily done by simply replacing FGFS scheduling in γ-FGFS with EDF scheduling when testing both for job acceptance feasibility and during time slot assignment. This is shown in

Algorithm 4 γ-EDF

1: A job (m, j) is generated at time $t_{m,j}$. Mobile m sends an upload request to the cloud server.
2: **if** $\mathcal{E}_{T,m} + e_{m,j}|_{x_{m,j}=0} < \gamma_m \, \mathcal{E}_{\max}$ **then**
3: $\quad x_{m,j} = 0$
4: **else**
5: $\quad$ Execute lines 2-16 in Algorithm 3
6: **end if**
7: Execute lines 7-13 in Algorithm 2
8: **return**

Algorithm 4 where the job is processed locally if the condition in line 2 is true, which is similar to γ-FGFS. Otherwise, line 5 uses EDF. The updates of γ_m in step 7 are the same as in γ-FGFS. The complexity of γ-EDF is the same as that for EDF.

3.6 Summary

In this chapter, four online scheduling algorithms were proposed based on closed-looped MCO. The computational complexity of each algorithm was also given. Before the performance of these schedulers is evaluated in Chapter 5, an open-loop MCO scheduling algorithm is given in Chapter 4 for comparison purposes, which highlights the benefits of closed-loop MCO over the open-loop approach.

Chapter 4

Open Loop Job Submission (OLJS)

4.1 Overview

To further evaluate the benefits of closed-loop cloud server scheduling in Chapter 3, we include comparisons where the cloud server processes jobs in first-come-first-served (FCFS) order and per-job submission feedback is not provided. Specifically, we investigate an *open loop job submission* model which is subject to statistical delay constraints without real-time information exchange.

4.2 OLJS Model

In OLJS, the users flow control their remote job submissions so that a given performance criterion can be met. In order to obtain energy fairness however, the flow control applied must be different for different user classes. For example, energy advantaged classes should typically be flow controlled more aggressively than those of who are energy disadvantaged. This differentiated flow control will increase the

rate at which energy disadvantaged users can offload, thus improving the min-max energy performance.

As noted in Section 2.2, it is not possible to meet hard delay constraints in the open-loop case, and therefore statistical delay guarantees must be considered instead. When mobile user requests arrive according to a Poisson process, for example, the total mean upload job submission rate must be below a certain threshold, defined by λ^*, in order to ensure that the probability that a job deadline is exceeded is maintained below a given threshold ε. To achieve the desired objective, the remote job submission rate of each user class is flow controlled using a job submission probability. When a mobile user from class m generates a job, it submits a request for cloud execution with probability ρ_m. In the case where the aggregate job arrivals from all mobile users of each class occur according to a Poisson arrival process, it is possible to analytically compute the value of λ^* so that the target value of ε is achieved. The case where jobs have fixed execution times is derived in the following development, however, the same procedure can be used in the case of general job execution time distributions.

Once the optimum value of λ^* has been determined, a linear program is used to compute the best values of ρ_m so that a min-max energy criterion is achieved among mobile users in different classes. This model then gives the optimum min-max energy assignment, and shows how well the best statically flow controlled scheduler can perform. It is used in Chapter 5 for comparisons with the other online schedulers.

We now use an analytic model as shown in Figure 4.1, in order to find the optimum job submission probability for each user class and the min-max energy consumption based on the OLJS model. For comparison with the other online scheduling algorithms, we consider the homogeneous case where: i) all the uploaded

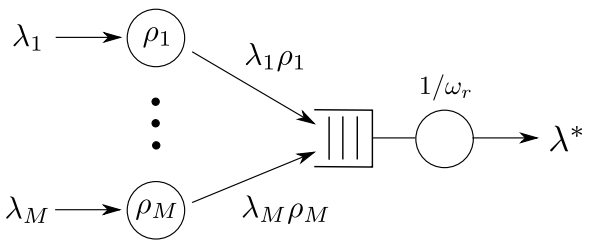

Figure 4.1: Analytic flow controlled job submission model

jobs have the same cloud server job service time, ii) jobs in mobile user class m are generated by a Poisson process with average rate λ_m, and, iii) for each job generated by a class m user, it is uploaded to the cloud server with probability ρ_m. Under these assumptions the aggregate mean job arrival rate at the cloud server is given by $\widetilde{\lambda} = \sum_m \rho_m \lambda_m$. This system can be modeled as an M/D/1 queue, whose waiting time distribution is given in (Erlang, 1909) and (Franx, 2001) by

$$\Pr\{\nu \leq \widehat{\nu}\} = \big(1 - \frac{\widetilde{\lambda}}{\omega_r}\big) \sum_{z=0}^{\lfloor \widehat{\nu}\omega_r \rfloor} \frac{[\widetilde{\lambda}(\frac{z}{\omega_r} - \widehat{\nu})]^z}{z!} e^{-\widetilde{\lambda}(\frac{z}{\omega_r} - \widehat{\nu})} \tag{4.1}$$

where ω_r is the job service rate at the cloud server, and $\lfloor \widehat{\nu}\omega_r \rfloor$ is the largest integer no greater than $\widehat{\nu}\omega_r$. Given that $\widehat{\nu}$ is the target job delay, we consider the tail distribution of delay, and define ε as the probability that the completion time of a job exceeds the required deadline, i.e.,

$$\Pr\{\nu \leq \widehat{\nu}\} \geq 1 - \varepsilon. \tag{4.2}$$

Denote the maximum value of $\widetilde{\lambda}$ that satisfies (4.2) as λ^*, which is the maximum uploaded traffic load from all mobile users. Note that the deterministic job service time assumption, i.e., assumption i), can be relaxed to a general service time distribution, thus modeling the cloud server as an M/G/1 queuing system. In this case, the job delay distribution can be obtained by numerically inverting the delay

probability generating function.

Once λ^* is found, the following optimization is used to find the optimum uploading probabilities for the user classes.

$$\begin{aligned} \underset{\rho_m}{\text{minimize}} \quad & \sum_{m\in\mathcal{M}} E_m && \text{(OLJS)} \\ \text{subject to} \quad & E_m \leq \widehat{E}, \quad \forall m \in \mathcal{M} && (4.3) \\ & \sum_{m\in\mathcal{M}} \rho_m \lambda_m \leq \lambda^* && (4.4) \\ & \rho_m \in [0,1], \quad \forall m \in \mathcal{M} && (4.5) \end{aligned}$$

where

$$E_m = (1-\rho_m)Q_m^l l_{m,j} + \rho_m(Q_m^t u_{m,j} + Q_m^r d_{m,j}). \tag{4.6}$$

The expression in (4.6) is based on (2.1) after taking the probabilistic upload decisions into consideration. As in (OPT), $\widehat{E}$ is a common upper bound on energy that is used to obtain the optimum min-max solution, and this is done using a binary search on $\widehat{E}$.

4.3 Closed Form Solution

In this part we show that the optimization of (OLJS) can be solved in closed form.

We define

$$E_m^r \triangleq Q_m^t u_{m,j} + Q_m^r d_{m,j}, \quad E_m^l \triangleq Q_m^l l_{m,j},$$

for all $m \in \mathcal{M}$, and (4.6) can be rewritten as

$$E_m = \rho_m E_m^r + (1 - \rho_m) E_m^l. \tag{4.7}$$

Since energy reduction is assumed to favour cloud execution, we have $E_m^r < E_m^l$. Without loss of generality, we sort all mobile user classes according to E_m^l as $E_1^l \leq E_2^l \leq \cdots \leq E_M^l$, and define $E_{\max}^r \triangleq \max_m\{E_m^r\}$. Also, note that the objective is min-max energy use, and therefore $\widehat{E} \in [E_{\max}^r, E_M^l]$. Below, we consider two different cases: $\widehat{E} = E_{\max}^r$ and $\widehat{E} > E_{\max}^r$.

Case 1: $\widehat{E} = E_{\max}^r$. This happens when the following conditions are satisfied:

$$\rho_m^* = 0, \forall\ m \text{ with } E_m^l \leq E_{\max}^r \tag{4.8}$$

$$\rho_m^* = \frac{E_m^l - E_{\max}^r}{E_m^l - E_m^r}, \forall\ m \text{ with } E_m^l > E_{\max}^r \tag{4.9}$$

$$\sum_m \rho_m^* \lambda_m \leq \lambda^*. \tag{4.10}$$

In this case, all user classes with $E_m^l > E_{\max}^r$ have $E_m = E_{\max}^r$, and other user classes have $E_m < E_{\max}^r$. The result in (4.9) can be proved by letting $E_m = \rho_m E_m^r + (1 - \rho_m) E_m^l \leq E_{\max}^r$.

This case is illustrated in Figure 4.2, which shows E_m versus value of ρ_m for five mobile user classes in the system. The minimum uploading probability is 0 for user class 1; for user classes 2, 3 and 5, ρ_m^* in (4.9) gives the minimum value for their uploading probabilities; and as a special case, the uploading probability for user class 4 is 1, since this is the class that has $E_{\max}^r$. If $\sum_m \rho_m \lambda_m < \lambda^*$, where $\rho_m \leq \rho_m^*$ for all m with $E_m^r < E_{\max}^r$, then user classes 1, 2, 3, and 5 are allowed to have higher

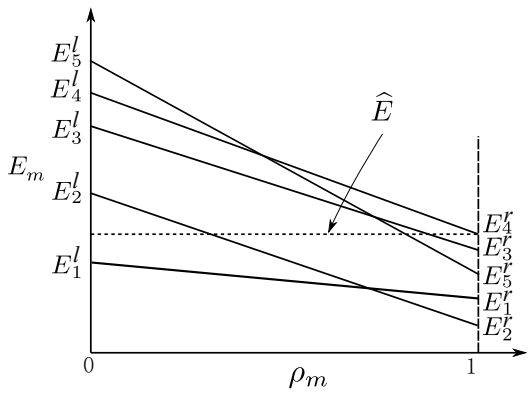

Figure 4.2: E_m vs. ρ_m where $\widehat{E} = E^r_{\max}$

uploading probabilities without increasing $\widehat{E}$, provided (4.10) is still satisfied.

Case 2: $\widehat{E} > E^r_{\max}$. Here the results in (4.8) and (4.9) do not always guarantee the condition in (4.10). In this case where the condition in (4.10) is not satisfied based on the the uploading probabilities in Case 1, higher min-max energy consumption, i.e., $\widehat{E} > E^r_{\max}$, is reached as shown in Figure 4.3, and the uploading probabilities are given below.

Without loss of generality, we assume that $E_1^l \leq E_2^l \leq \cdots \leq E_{k-1}^l \leq \widehat{E} \leq E_k^l \leq \cdots \leq E_M^l$, where $1 \leq k \leq M$, then the following conditions should be satisfied:

$$\rho_m^* = 0, \ \forall \ m \leq k-1 \tag{4.11}$$

$$\rho_k^* = \frac{\lambda^* - \sum_{m=k}^{M} \frac{(E_m^l - E_k^l)\lambda_m}{E_m^l - E_m^r}}{\sum_{m=k}^{M} \frac{(E_k^l - E_k^r)\lambda_m}{E_m^l - E_m^r}} \tag{4.12}$$

$$\rho_m^* = \frac{(E_k^l - E_k^r)\rho_k^* + (E_m^l - E_k^l)}{E_m^l - E_m^r}, \ \forall \ m \geq k+1 \tag{4.13}$$

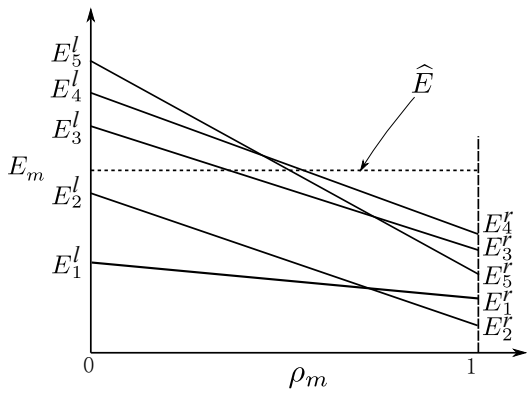

Figure 4.3: E_m vs. ρ_m where $\widehat{E} > E^r_{\max}$ and $k = 3$

where

$$k = \arg\min_i \left\{ \frac{\lambda^* - \sum_{m=i}^{M} \frac{(E_m^l - E_i^l)\lambda_m}{E_m^l - E_m^r}}{\sum_{m=i}^{M} \frac{(E_i^l - E_i^r)\lambda_m}{E_m^l - E_m^r}} \geq 0 \right\}. \tag{4.14}$$

Below we prove this result. Note that all the uploading probabilities should satisfy $\sum_m \rho_m \lambda_m = \lambda^*$. For all $m \leq k$, $\rho_m = 0$ and $E_m \leq E_k$; and for all $m \geq k$, $\rho_m > 0$ and $E_m = E_k$. Considering $m \geq k$, we have

$$E_m^r \rho_m + E_m^l (1 - \rho_m) = E_k^r \rho_k + E_k^l (1 - \rho_k) \tag{4.15}$$

from which we can solve

$$\rho_m = \frac{(E_k^l - E_k^r)\rho_k + (E_m^l - E_k^l)}{E_m^l - E_m^r}.$$

Next, substituting all ρ_m's into $\sum_{m=k}^{M} \rho_m \lambda_m = \lambda^*$, we have

$$\sum_{m=k}^{M} \frac{(E_m^l - E_k^l)\lambda_m}{E_m^l - E_m^r} + \sum_{m=k}^{M} \frac{(E_k^l - E_k^r)\lambda_m}{E_m^l - E_m^r} \rho_k = \lambda^*,$$

and then

$$\rho_k^* = \frac{\lambda^* - \sum_{m=k}^{M} \frac{(E_m^l - E_k^l)\lambda_m}{E_m^l - E_m^r}}{\sum_{m=k}^{M} \frac{(E_k^l - E_k^r)\lambda_m}{E_m^l - E_m^r}}. \tag{4.16}$$

We can then find ρ_m for all $m \geq k$ by substituting ρ_k^* in (4.16) into (4.15), and further prove (4.13). With this, the min-max energy consumption is given as

$$\widehat{E} = E_k^r \rho_k^* + E_k^l (1 - \rho_k^*).$$

4.4 Summary

In this chapter, we introduced OLJS scheduling, which operates without real time interaction between the mobile users and the cloud server. Notably, the OLJS scheduler operates in a straightforward manner using FCFS where any new job is placed at the back of the queue, thus the complexity is $\mathcal{O}(1)$.

Simulation results are presented in Chapter 5, which demonstrate the performance differences between OLJS and closed-loop MCO based algorithms designed in Chapter 3.

Chapter 5

Performance Evaluation

5.1 Overview

In this chapter we present a variety of results that evaluate the performance of the proposed schedulers. Rather than focusing on any specific application, we instead explore a wide range of parameter settings that are selected to illustrate various properties of the online schedulers. The simulation settings are first given, followed by numerical results with detailed discussions.

5.2 Simulation Settings

In the initial set of results, we consider a set of mobile users where the jobs to be submitted arrive according to a Poisson process, and the job execution is identical. Each job is processed using either local or cloud server execution. The default simulation parameter settings are given in Table 5.1.

The different mobile user classes have wireless uplink transmission data rates that

Table 5.1: Default simulation parameter settings

$l_{m,j}$	267 (time slots)
$r_{m,j}$	5 (time slots)
Q_m^l/Q_m^t	3

range over a factor of about 25 when comparing the lowest and the highest values. In the results to be shown, we incorporate the job output download latency $d_{m,j}$ into that of the uplink, which can be done without any loss in generality (Zhang *et al.*, 2013). The delay constraint $D_{m,j}$ is set randomly between 1 and 1.5 times $l_{m,j}$ (i.e., between 267 and 400 time slots) after $t_{m,j}$, so that local execution is always a feasible option. Also, the maximum energy consumption plotted is normalized to the total number of jobs over all mobile user classes.

5.3 Numerical Results with $M = 2$

We first consider the case with $M = 2$ mobile user classes, and the ratio of uplink transmission rates between the two classes is exactly 25. Figure 5.1 shows the maximum energy consumption of each online scheduler, where the job generation rate is in numbers of jobs per time slot. It can be seen that as the job generation rate increases, the workload is heavier, and more jobs compete for the cloud server resources. As a result, the difference between the amount requested for uploading and the amount that can be accepted increases, and the maximum energy consumption goes up. As discussed in Section 2.3, OPT is a strict lower bound on maximum energy for any causal online scheduler. It can be seen in Figure 5.1 that the best of the online schedulers ranges from zero to about 5% above the optimum bound. Note that when the job generation rate is low enough, all of the algorithms

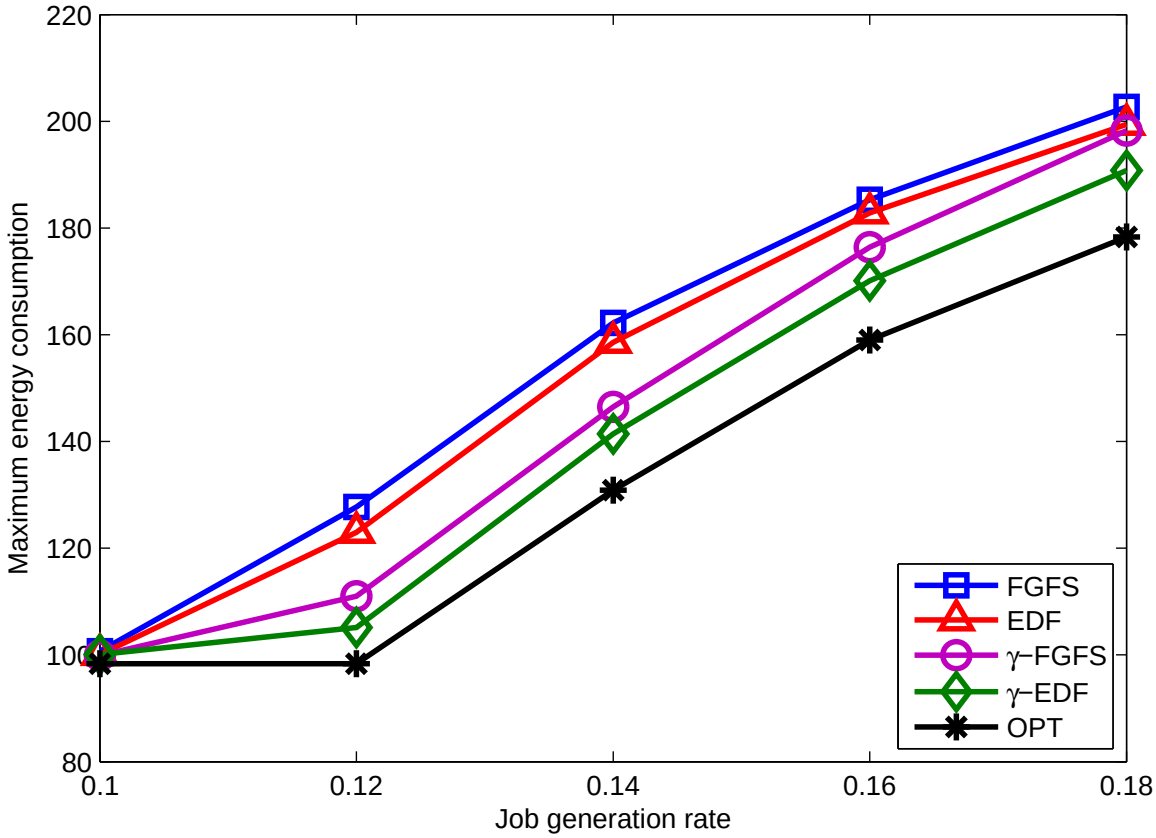

Figure 5.1: Maximum energy consumption vs. job generation rate ($M = 2$)

converge to the bound since, at that point, they all permit unrestricted job offloading.

Figure 5.1 also shows the general trend in the relative algorithm performance, i.e., the γ-based algorithms tend to outperform the others due to their min-max energy focus. Similarly, EDF tends to perform better than FGFS. This is because EDF can better accommodate jobs with more strict latency requirements, which helps to admit more offloading. However, in this example, where the user deadlines are homogeneous, the performance between the algorithms is fairly modest. The maximum spread in min-max energy performance between the algorithms is about

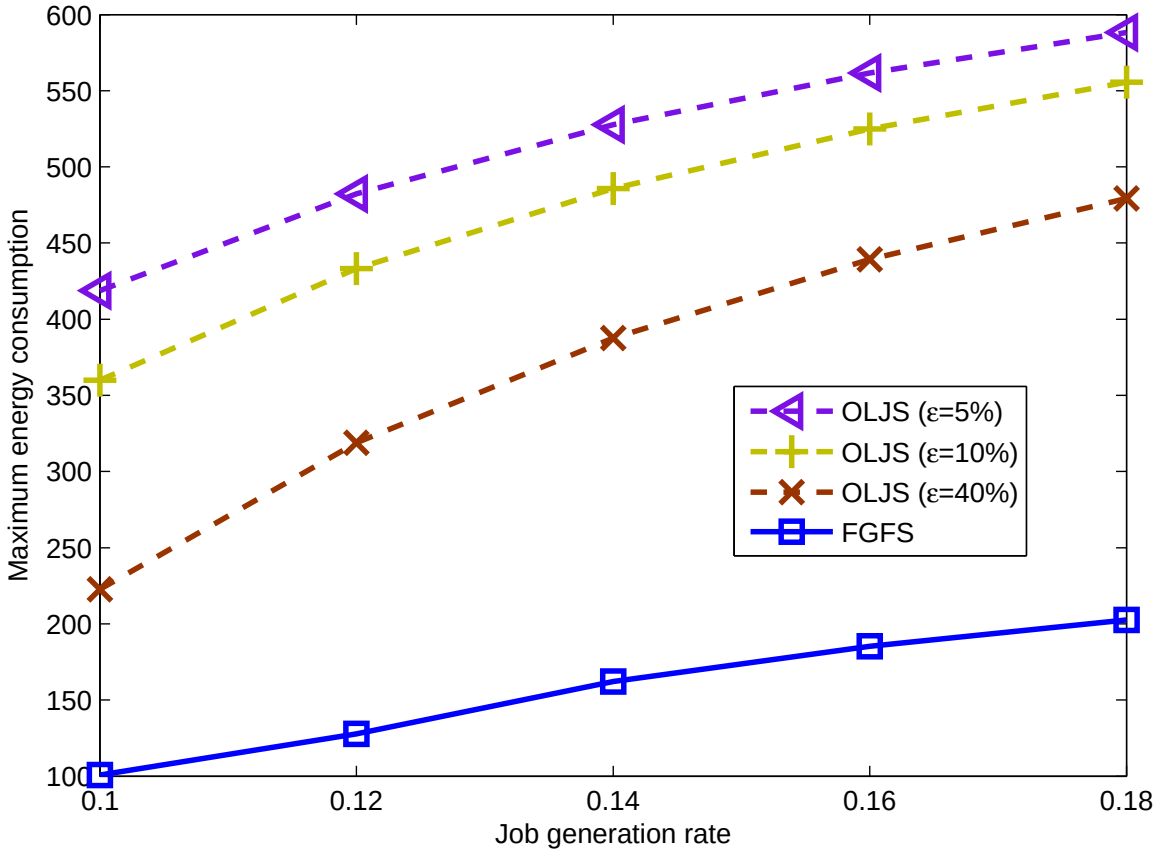

Figure 5.2: Energy performance of OLJS ($M = 2$)

30%, which is not completely insignificant. Figure 5.1 clearly shows that the γ-FGFS and γ-EDF schedulers can achieve lower energy consumption than straight EDF, which demonstrates that balancing the energy consumption of different user classes plays a more significant role, leading to improved min-max energy fairness.

In Figure 5.2 we show an example of comparisons using OLJS scheduling. Recall that in OLJS, deadline constraints are satisfied statistically using the ε probability, and Figure 5.2 shows the energy performance with ε set to 5%, 10% and 40%. As the value of ε increases, the maximum energy consumption decreases due to a larger tolerance on job deadline constraint violation. However, even though the values of ε

are relatively large, the min-max energy performance of OLJS is poor compared to the worst online algorithm, i.e., FGFS. With a ε of 5% for example, the min-max delay ranges from about 300% to 400% of that obtained using simple FGFS scheduling. For this reason, OLJS is a poor option when constraints on delay are desired. This confirms the usefulness of real-time information exchange between the mobile users and the cloud server.

5.4 Numerical Results with $M = 4$

To further illustrate various aspects of online algorithm energy fairness, in the rest of the simulation results we consider a scenario with $M = 4$, where one mobile user class has poor channel conditions, referred to as the P class, and the other three classes have good channel conditions, referred to as the G classes. The uplink transmission data rate of three G classes is set to be 22, 25 and 28 times that of the P class, respectively. At the same time, the delay for P class and G classes is set to 300 and 400 time slots, so that the P class is further constrained from a deadline point of view. The job generation rate is fixed to be 0.08 jobs per time slot.

Figure 5.3 shows the maximum energy consumption as the uplink transmission energy per time slot increases. Obviously, with greater values, the users consume more energy for offloading, which offsets the effectiveness of energy reduction with the help of cloud execution. For this reason, the gap between FGFS and γ-FGFS, as well as that between EDF and γ-EDF, gradually becomes smaller in the sense that the margin of energy saving with which γ-FGFS and γ-EDF are able to play, is reduced. This makes them less useful as computation offloading becomes less attractive. However, it can be seen that at the left hand side of the graph, the performance improvements

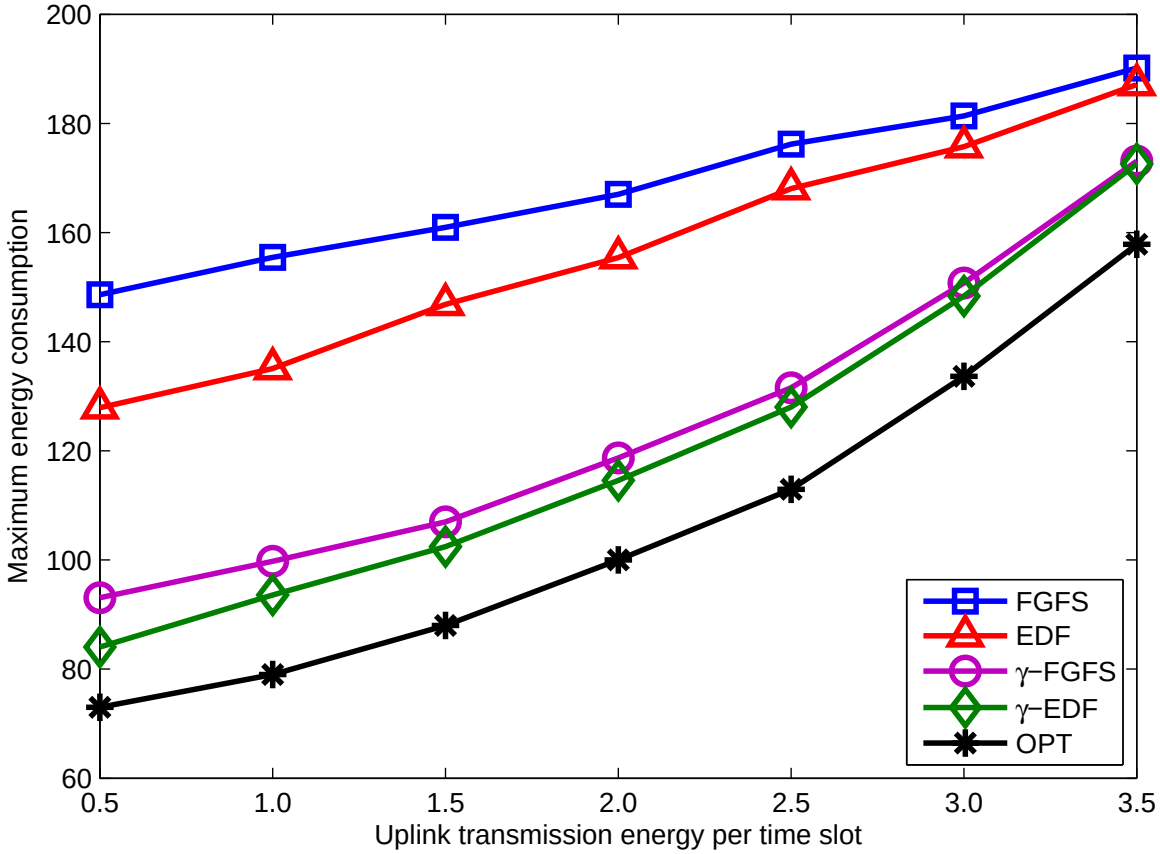

Figure 5.3: Maximum energy consumption vs. uplink transmission energy per time slot ($M = 4$)

associated with the γ-based algorithms are significant. The maximum energy of EDF compared to γ-EDF is over 50% and compared to FGFS the increase is almost a factor of 80%. These improvements decrease significantly as the transmission energy increases since the energy performance is dominated by this large uploading cost.

Figure 5.4 shows the energy consumption trends with respect to the time for processing one job unit using cloud execution. Because of the deadline constraints, the number of jobs that can be accepted by the cloud server decreases as the efficiency of cloud execution goes down. When the efficiency is sufficiently high, the cloud

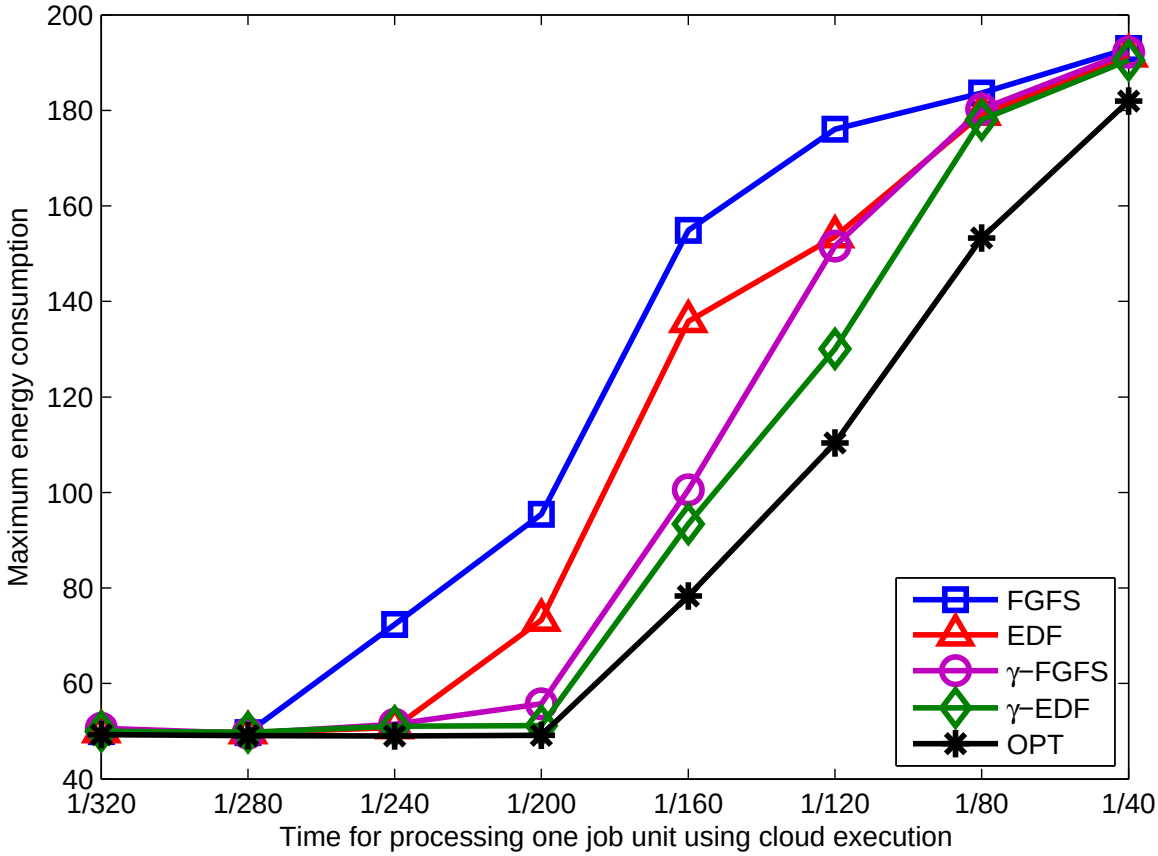

Figure 5.4: Maximum energy consumption vs. time for processing one job unit using cloud execution ($M = 4$)

server loading decreases to the point where all jobs can be uploaded without any deadline violation. For this reason, the algorithms all converge for low cloud server processing times. Similarly, when processing times increase, eventually all P user jobs are unable to offload, and all execution occurs locally. This results in the convergence of the algorithm performance at the right hand side of the graph. However, for values between these two extremes, the schedulers show significant differences in maximum energy performance. As before, the γ-EDF scheduler performs best overall. For example, EDF can be seen to be about 40% higher than that of γ-EDF near the

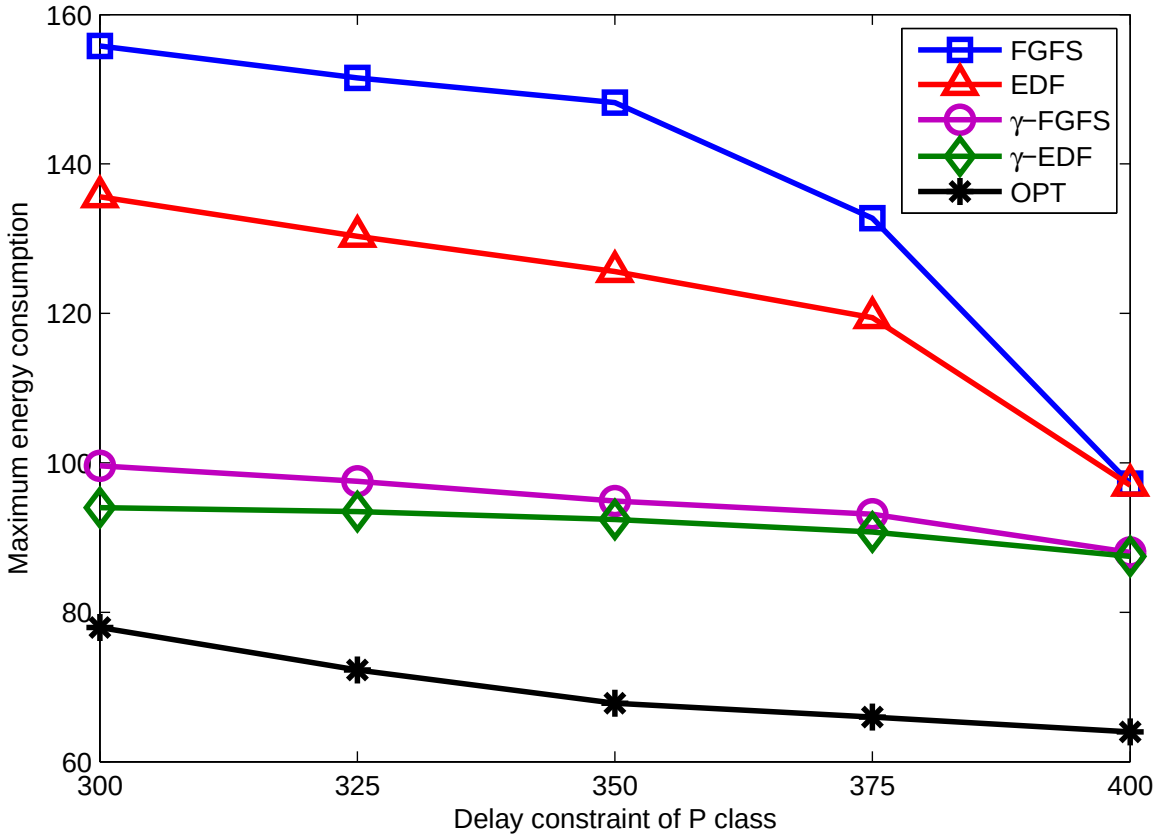

Figure 5.5: Maximum energy consumption vs. P class delay (G class delay is 400, $M = 4$)

midpoint of the extreme values. Also, FGFS is clearly the most sensitive to the cloud execution efficiency because of the lack of scheduling flexibility.

In order to study how delay constraints influence scheduler performance, the results for different combinations of delay constraints are given in Figures 5.5 and 5.6. In Figure 5.5, the delay constraint for the G classes is fixed to 400 time slots. As the delay constraint of the P class becomes more loose, the cloud server becomes less apt to reject P class jobs due to deadline violations. When this happens, the maximum energy consumption decreases. When the delay constraints of the P and G classes are close, however, the difference between FGFS and EDF almost

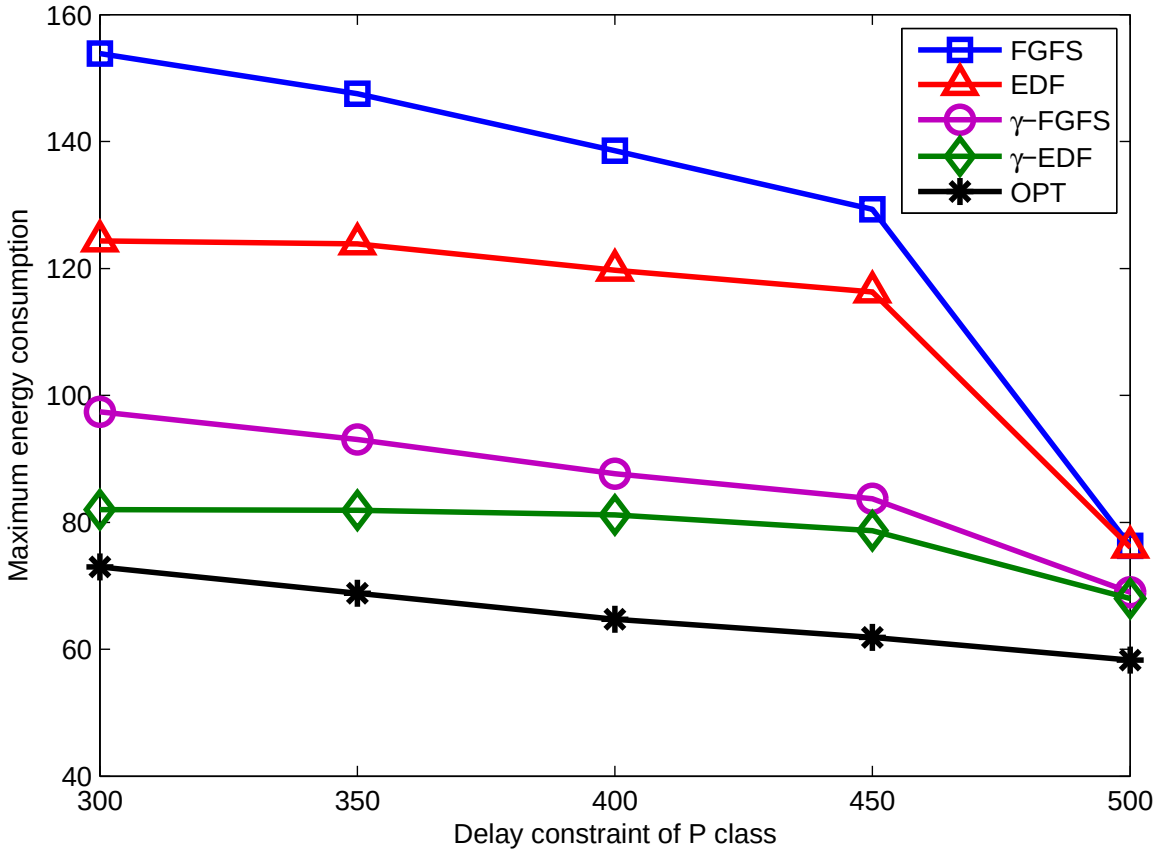

Figure 5.6: Maximum energy consumption vs. P class delay (G class delay is 500, $M = 4$)

disappears, which further benefits the P class users because their disadvantage of more stringent delay constraints no longer exists.

Figure 5.6 illustrates the case where the delay constraint of the G classes is set to 500 time slots. It can be seen that the curves evolve in a fashion similar to those of Figure 5.5 while performing even better by achieving a lower level of maximum energy consumption. Note that γ-FGFS and γ-EDF, which explicitly consider energy performance, track the OPT bound more closely, compared to that of the other algorithms whose performance is significantly worse. In particular, γ-EDF can

improve the maximum energy consumption significantly. In this figure, the decrease is by almost 50% compared with FGFS.

5.5 Summary

In this chapter, we presented a number of simulation results to evaluate the performance of all schedulers. It was shown that closed-loop MCO algorithms outperform the open-loop approach thanks to the real-time information exchange. Also, γ-EDF achieves the best level of energy fairness among online scheduling algorithms.

Chapter 6

Conclusions and Future Works

In this thesis we have considered a system where mobile users reduce their energy consumption using computation offloading to a remote cloud server. Since job processing is subject to deadline completion constraints, this type of system can suffer from energy unfairness when, for example, some mobile users have unfavourable channel conditions compared to the others. Scheduling algorithms were proposed that can compensate for this unfairness by dynamic scheduling at the cloud server. An optimum offline scheduler was first proposed that solves an integer linear program based on a min-max energy objective.

A variety of online scheduling algorithms were then introduced. FGFS was first proposed that performs job acceptance and scheduling on the basis of FGFS scheduling. EDF scheduling was then used to test for job scheduling feasibility. Two modified versions, referred to as γ-FGFS and γ-EDF, were proposed where the acceptance of job submission is subject to an energy threshold test. To highlight the importance of information exchange, we also considered an OLJS model that flow controls the job submission but leads to far greater energy consumption.

Various performance results were presented that show the improvements in energy fairness possible with the proposed schedulers. Our results showed that overall, γ-EDF outperforms all other online algorithms. In particular, the maximum energy consumption of γ-EDF can be less than 5% above the optimum bound in the homogeneous deadline case, while γ-EDF is able to significantly reduce the maximum energy consumption by approximately 50% compared with FGFS when the network includes users with poor channel conditions and stringent deadline constraints.

The work in this thesis is based on binary job partitions. For future work, multiple job partitions may be taken into account, such that computation offloading decisions can be made in a more flexible fashion for different types of mobile applications. Correspondingly, current scheduling algorithms can be effectively extended to the ones that accommodate multiple partitions. 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. As a further step, energy fairness among mobile users can possibly be achieved by joint channel and cloud server scheduling, where the allocation of wireless channels is taken into account as another way of helping to compensate for disadvantaged mobile users.

References

Abolfazli, S., Sanaei, Z., Gani, A., Xia, F., and Yang, L. T. (2014). Rich Mobile Applications: Genesis, Taxonomy, and Open Issues. *Journal of Network and Computer Applications*, **40**, 345–362.

Achary, R., Vityanathan, V., Raj, P., and Nagarajan, S. (2015). Dynamic Job Scheduling Using Ant Colony Optimization for Mobile Cloud Computing. In *Intelligent Distributed Computing*, pages 71–82.

Barbarossa, S., Sardellitti, S., and Lorenzo, P. D. (2013). Joint Allocation of Computation and Communication Resources in Multiuser Mobile Cloud Computing. *2013 IEEE 14th Workshop on Signal Processing Advances in Wireless Communications*, pages 26–30.

Chen, X. (2015). Decentralized Computation Offloading Game for Mobile Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems*, **26**(4), 974–983.

Chun, B.-G. and Maniatis, P. (2009). Augmented Smartphone Applications Through Clone Cloud Execution. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, pages 8–11.

Chun, B. G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. (2011). Clonecloud: Elastic Execution between Mobile Device and Cloud. *6th ACM Conference on Computer Systems*, pages 301–314.

Cisco (2015). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update. *http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html*.

Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. (2010). MAUI: Making Smartphones Last Longer with Code Offload. *8th ACM International Conference on Mobile Systems, Applications, and Services*, pages 49–62.

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.

Doukas, C., Pliakas, T., and Maglogiannis, I. (2010). Mobile Healthcare Information Management Utilizing Cloud Computing and Android OS. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 1037–1040.

Elgazzar, K., Martin, P., and Hassanein, H. (2015). Cloud-Assisted Computation Offloading to Support Mobile Services. *IEEE Transactions on Cloud Computing*.

Erlang, A. K. (1909). The Theory of Probabilities and Telephone Conversations. *Nyt Tidsskrift for Matematik*, **20**(B), 33–39.

Fan, X., Cao, J., and Mao, H. (2011). A Survey of Mobile Cloud Computing. *ZTE Communications*, **9**(1), 4–8.

Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile Cloud Computing: A Survey. *Future Generation Computer Systems*, **29**(1), 84–106.

Ferzli, R. and Khalife, I. (2011). Mobile Cloud Computing Educational Tool for Image/Video Processing Algorithms. In *2011 IEEE Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop*, pages 529–533.

Franx, G. J. (2001). A Simple Solution for the M/D/1 Waiting Time Distribution. *Operations Research Letters*, **29**(5), 221–229.

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

Guan, L., Ke, X., Song, M., and Song, J. (2011). A Survey of Research on Mobile Cloud Computing. In *Proceedings of the 2011 10th IEEE/ACIS International Conference on Computer and Information Science*, pages 387–392.

Huang, D., Wang, P., and Niyato, D. (2012). A Dynamic Offloading Algorithm for Mobile Computing. *IEEE Transactions on Wireless Communications*, **11**(6), 1991–1995.

Huang, D., Zhu, C., Zhang, H., and Liu, X. (2014). Resource Intensity Aware Job Scheduling in A Distributed Cloud. *China Communications*, **11**(14), 175–184.

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*, number 6.

Jeffay, K., Stanat, D. F., and Martel, C. U. (1991). On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. In *Twelfth Real-Time Systems Symposium*, pages 129–139.

Kaewpuang, R., Niyato, D., Wang, P., and Hossain, E. (2013). A Framework for Cooperative Resource Management in Mobile Cloud Computing. *IEEE Journal on Selected Areas in Communications*, **31**(12), 2685–2700.

Kemp, R., Palmer, N., Kielmann, T., and Bal, H. (2012). Cuckoo: A Computation Offloading Framework for Smartphones. In *Mobile Computing, Applications, and Services*, pages 59–79.

Khan, A., Othman, M., Madani, S., and Khan, S. (2014). A Survey of Mobile Cloud Computing Application Models. *IEEE Communications Surveys & Tutorials*, **16**(1), 393–413.

Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. (2012). ThinkAir: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading. *IEEE INFOCOM*, pages 945–953.

Kovachev, D., Cao, Y., and Klamma, R. (2011). Mobile Cloud Computing: A Comparison of Application Models. *arXiv preprint arXiv:1107.4940*.

Kruk, L., Lehoczky, J., Shreve, S., Yeung, S.-N., *et al.* (2004). Earliest-Deadline-First Service in Heavy-Traffic Acyclic Networks. *The Annals of Applied Probability*, **14**(3), 1306–1352.

Kumar, K. and Lu, Y. H. (2010). Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? *Computer*, **43**(4), 51–56.

Lei, L., Zhong, Z., Zheng, K., Chen, J., and Meng, H. (2013). Challenges on Wireless Heterogeneous Networks for Mobile Cloud Computing. *IEEE Wireless Communications*, **20**(3), 34–44.

Li, W. (2006). *Group-EDF: A New Approach and an Efficient Non-Preemptive Algorithm for Soft Real-time Systems*. Ph.D. thesis, University of North Texas.

Liang, H., Huang, D., and Peng, D. (2012). On Economic Mobile Cloud Computing Model. In *Mobile Computing, Applications, and Services*, pages 329–341.

Lin, X., Wang, Y., Xie, Q., and Pedram, M. (2014). Energy and Performance-Aware Task Scheduling in A Mobile Cloud Computing Environment. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 192–199.

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. *IEEE Wireless Communications*, **20**(3), 14–22.

Ma, X., Zhao, Y., Zhang, L., Wang, H., and Peng, L. (2013). When Mobile Terminals Meet the Cloud: Computation Offloading as the Bridge. *IEEE Network*, **27**(5), 28–33.

Maguluri, S. T. and Srikant, R. (2013). Scheduling Jobs with Unknown Duration in Clouds. In *2013 Proceedings IEEE INFOCOM*, pages 1887–1895.

Magurawalage, C. M. S., Yang, K., Hu, L., and Zhang, J. (2014). Energy-Efficient and Network-Aware Offloading Algorithm for Mobile Cloud Computing. *Computer Networks*, **74**, 22–33.

March, V., Gu, Y., Leonardi, E., Goh, G., Kirchberg, M., and Lee, B. S. (2011). μCloud: Towards A New Paradigm of Rich Mobile Applications. *Procedia Computer Science*, **5**, 618–624.

Miettinen, A. P. and Nurminen, J. K. (2010). Energy Efficiency of Mobile Clients in Cloud Computing. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, number 4, pages 1–7.

Namboodiri, V. and Ghose, T. (2012). To Cloud or Not to Cloud: A Mobile Device Perspective on Energy Consumption of Applications. In *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–9.

Niyato, D., Hossain, E., and Han, Z. (2009). Dynamics of Multiple-Seller and Multiple-Buyer Spectrum Trading in Cognitive Radio Networks: A Game-Theoretic Modeling Approach. *IEEE Transactions on Mobile Computing*, **8**(8), 1009–1022.

Oberheide, J., Cooke, E., and Jahanian, F. (2008). CloudAV: N-Version Antivirus in the Network Cloud. In *USENIX Security Symposium*, pages 91–106.

Othman, M. and Hailes, S. (1998). Power Conservation Strategy for Mobile Computers Using Load Sharing. *ACM SIGMOBILE Mobile Computing and Communications Review*, **2**(1), 44–51.

Parekh, A. K. and Gallager, R. G. (1993). A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: the Single-Node Case. *IEEE/ACM Transactions on Networking*, **1**(3), 344–357.

Pendyala, V. S. and Holliday, J. (2010). Performing Intelligent Mobile Searches in

the Cloud Using Semantic Technologies. In *2010 IEEE International Conference on Granular Computing*, pages 381–386.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons.

Rong, P. and Pedram, M. (2003). Extending the Lifetime of A Network of Battery-Powered Mobile Devices by Remote Processing: A Markovian Decision-Based Approach. In *Proceedings of the 40th annual Design Automation Conference*, pages 906–911.

Rudenko, A., Reiher, P., Popek, G. J., and Kuenning, G. H. (1998). Saving Portable Computer Battery Power through Remote Process Execution. *Journal of ACM SIGMOBILE on Mobile Computing and Communications Review*, **2**(1), 19–26.

Rudenko, A., Reiher, P., Popek, G. J., and Kuenning, G. H. (1999). The Remote Processing Framework for Portable Computer Power Saving. In *Proceedings of the 1999 ACM Symposium on Applied Computing*, pages 365–372.

Ryzhov, I., Frazier, P. I., and Powell, W. B. (2014). A New Optimal Stepsize for Approximate Dynamic Programming. *IEEE Transactions on Automatic Control*, **60**(3), 743–758.

Sahlins, M. D. and Service, E. R. (1988). *Evolution and Culture*, volume 197. University of Michigan Press.

Sardellitti, S., Scutari, G., and Barbarossa, S. (2014). Joint Optimization of Radio and Computational Resources for Multicell Mobile Cloud Computing. In *2014*

IEEE 15th International Workshop on Signal Processing Advances in Wireless Communications, pages 354–358.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, **8**(4), 14–23.

Schwiegeishohn, U. and Yahyapour, R. (1998). Improving First-Come-First-Serve Job Scheduling by Gang Scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 180–198.

Song, J., Cui, Y., Li, M., Qiu, J., and Buyya, R. (2014). Energy-Traffic Tradeoff Cooperative Offloading for Mobile Cloud Computing. In *2014 IEEE 22nd International Symposium of Quality of Service*, pages 284–289.

Tang, W.-T., Hu, C.-M., and Hsu, C.-Y. (2010). A Mobile Phone Based Homecare Management System on the Cloud. In *2010 3rd International Conference on Biomedical Engineering and Informatics*, pages 2442–2445.

Vallina-Rodriguez, N. and Crowcroft, J. (2013). Energy Management Techniques in Modern Mobile Handsets. *IEEE Communications Surveys & Tutorials*, **15**(1), 179–198.

Wan, J., Zou, C., Ullah, S., Lai, C.-F., Zhou, M., and Wang, X. (2013). Cloud-Enabled Wireless Body Area Networks for Pervasive Healthcare. *IEEE Network*, **27**(5), 56–61.

Wu, Y., Zhang, Z., Wu, C., Li, Z., and Lau, F. (2013). CloudMoV: Cloud-Based Mobile Social TV. *IEEE Transactions on Multimedia*, **15**(4), 821–832.

Xian, C., Lu, Y.-H., and Li, Z. (2007). Adaptive Computation Offloading for Energy Conservation on Battery-Powered Systems. In *2007 International Conference on Parallel and Distributed Systems*, pages 1–8.

Xu, Y. and Wen, S. (2013). A Survey of Mobile Cloud Computing for Rich Media Applications. *IEEE Wireless Communications*, **20**(3), 46–53.

Yang, X., Pan, T., and Shen, J. (2010). On 3G Mobile E-Commerce Platform Based on Cloud Computing. In *2010 3rd IEEE International Conference on Ubi-Media Computing*, pages 198–201.

Yue, J., Yang, B., Chen, C., and Guan, X. (2014a). Chasing the Most Popular Video: An Evolutionary Video Clip Selection. *IEEE Communications Letters*, **18**(5), 781–784.

Yue, J., Zhao, D., and Todd, T. D. (2014b). Cloud Server Job Selection and Scheduling in Mobile Computation Offloading. In *2014 IEEE Global Communications Conference*, pages 4990–4995.

Yue, J., Yang, B., Chen, C., Guan, X., and Zhang, W. (2014c). Femtocaching in Video Content Delivery: Assignment of Video Clips to Serve Dynamic Mobile Users. *Computer Communications*, **51**, 60–69.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud Computing: State-of-the-Art and Research Challenges. *Journal of Internet Services and Applications*, **1**(1), 7–18.

Zhang, W., Wen, Y., Guan, K., Kilper, D., Luo, H., and Wu, D. O. (2013). Energy-Optimal Mobile Cloud Computing under Stochastic Wireless Channel. *IEEE Transactions on Wireless Communications*, **12**(9), 4569–4581.

Zhao, W., Sun, Y., and Dai, L. (2010). Improving Computer Basis Teaching Through Mobile Communication and Cloud Computing Technology. In *2010 3rd International Conference on Advanced Computer Theory and Engineering*, volume 1, pages 452–454.

Zhu, C., Leung, V. C., Hu, X., Shu, L., and Yang, L. T. (2013). A Review of Key Issues that Concern the Feasibility of Mobile Cloud Computing. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 769–776.