Channel and Server Scheduling for Energy-Fair Mobile Computation Offloading

CHANNEL AND SERVER SCHEDULING FOR ENERGY-FAIR MOBILE COMPUTATION OFFLOADING

BY

JONATHAN MOSCARDINI, B.Eng.Mgmt.

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 (2016)	McMaster University
(Electrical & Computer Engineering)	Hamilton, Ontario, Canada

TITLE:	Channel and Server Scheduling for Energy-Fair Mobile Computation Offloading
AUTHOR:	Jonathan Moscardini B.Eng.Mgmt. (Computer Engineering and Management) McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. Terence D. Todd
NUMBER OF PAGES:	xiii, 62

To my family, for their infinite love and support; and to my mom in particular, in recognition of her own academic achievement

Abstract

This thesis investigates energy fairness in an environment where multiple mobile cloud computing users are attempting to utilize both a shared channel and a shared server to offload jobs to remote computation resources, a technique known as mobile computation offloading. This offloading is done in an effort to reduce energy consumption at the mobile device, which has been demonstrated to be highly effective in previous work. However, insufficient resources are available for all mobile devices to offload all generated jobs due to constraints at the shared channel and server. In addition to these constraints, certain mobile devices are at a disadvantage relative to others in their achievable offloading rate. Hence, the shared resources are not necessarily shared fairly, and an effort must be made to do so.

A method for improving offloading fairness in terms of total energy is derived, in which the state of the queue of jobs waiting for offloading is evaluated in an online fashion, at each job arrival, in order to inform an offloading decision for that newest arrival; no prior state or future predictions are used to determine the optimal decision. This algorithm is evaluated by comparing it on several criteria to standard scheduling methods, as well as to an optimal offline (i.e., non-causal) schedule derived from the solution of a min-max energy integer linear program. Various results derived by simulation demonstrate the improvements in energy fairness achieved.

Acknowledgements

The conclusion of this thesis marks the end of a continuous 8 years of study at McMaster for me; the end of such a lengthy stay in any one place is bound to evoke numerous emotions. Chief among them here may be elation, immense relief, or perhaps a sense of imminent freedom, but my departure also brings many fond recollections of my time here and of the people who have helped me achieve what I have.

Above all else I give my sincere thanks to my supervisor, Dr. Terry Todd, for his endless patience, encouragement, advice and direction as I navigated my first research endeavour. This was at many points a difficult process for me and I would not have seen its successful completion without his efforts, or his anecdotes. I also thank Dr. Dongmei Zhao for her input and assistance over the past two years.

At times gratitude was not the emotion I wished to express to Dr. Steve Hranilovic, also of the Electrical and Computer Engineering department at McMaster, for encouraging me to embark on this journey as I debated the merits of a graduate degree two years ago. But it is certainly gratitude I express to him now, as without that encouragement I would likely not be writing this thesis today.

I would of course like to thank my parents for their unwavering belief in my abilities, their endless support and encouragement through the highs and the lows, and their exceedingly affordable food service and rent. I would also like to thank my cousin, Heather Ruhl, for her support in the completion of this thesis, taking time out of her vacation to be interested in my efforts.

Finally I would like to thank the family, friends, fellow students, lab colleagues past and present, and faculty who have helped me through nearly a decade of learning, inside and outside of the classroom. I can only hope that the people I encounter on my journey moving forward are as wonderful as you have been, because I couldn't have done it without you.

Notation

$\mathcal{M}$	set of mobile device classes
M	number of mobile device classes
$\mathcal{J}$	set of jobs
J	number of jobs
λ_m	arrival rate for job class m
$\mathcal{T}$	set of all time slots
$\mathcal{T}_{m,j}$	set of time slots usable for job m, j
T	number of time slots
$a_{m,j}$	arrival time of job (m, j)
$D_{m,j}$	deadline of job m, j
$z_{m,j,t}$	channel time slot selection variable for job m, j at time slot t
$y_{m,j,t}$	server time slot selection variable for job m, j at time slot t
$x_{m,j,p}$	partition selection variable for partition p, job m, j
$p_{m,j}$	a partitioning option with index p for job m, j
$P_{m,j}$	number of possible partitioning options
$l_{m,j,p}$	local processing units required for job m, j at partition $p_{m,j}$
$l_{m,j,1}$	local units required for local-only processing

$u_{m,j,p}$	transmission slots required for job m, j at partition $p_{m,j}$
$r_{m,j,p}$	remote server time slots required for job m, j at partition $p_{m,j}$
U_m	uploading rate for device class m
R_m	remote processing rate for device class m
$Q^l_{m,j,p}$	local processing per-unit energy
$Q^r_{m,j,p}$	remote transmission per-unit energy
$\mathcal{E}^l_{m,j,p}$	local processing energy for job m, j at partition $p_{m,j}$
$\mathcal{E}^u_{m,j,p}$	transmission energy for job m, j at partition $p_{m,j}$
$\mathcal{E}_{m,j,p}$	total processing energy for job m, j at partition $p_{m,j}$
$\mathcal{E}_{m,j}$	total processing energy for job m, j at the final selected partition
G	users/classes with good channel conditions
B	users/classes with poor channel conditions
u_G	upload rate for users with good channel conditions
u_B/u_G	upload rate for users with poor channel conditions
$\lambda_B/(\lambda_B + \lambda_G)$	arrival rate of users with poor channel conditions
C_q	AQE queue size cutoff threshold
$\Delta_{\mathcal{E}}$	difference in per-job energy between compared classes

Abbreviations

AQE	Average Queue Energy Prioritization
EDF	Earliest Deadline First
FCFS	First Come First Served
FGFS	First Generated First Served
ILP	Integer Linear Program
JIT	just-in-time
MCC	Mobile Cloud Computing
OS	operating system
VM	virtual machine

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile devices, and smartphones in particular, are becoming increasingly ubiquitous. The GSM Alliance expects the number of smartphones worldwide to increase to 5.6 billion, up from 3.2 billion in 2015 (George *et al.*, 2016). As mobile device usage grows, mobile device utilization of cloud computing grows with it; cloud applications are expected to account for 90% of worldwide mobile data traffic by 2019 (Cisco, 2015). And with that growth, consumer interest in improved battery life remains strong (Arthur, 2014).

This usage of cloud computing by mobile devices is known as mobile cloud computing (MCC). Sanaei *et al.* (2014) define it as "a rich mobile computing technology that leverages... varied clouds and network technologies toward unrestricted functionality, storage, and mobility to serve a multitude of mobile devices anywhere, anytime". The use of these resources can offer significant benefits in energy consumption, computing performance, and storage capacity, as processing and data storage are shifted from the mobile device to cloud servers with significantly greater resources.

1.1 Overview

To suggest offloading a task to a remote resource implies that the resource exists and is available. Fortunately, there are many examples of cloud computing resources which are available to mobile devices. These remote resources can take several forms, from dedicated cloud computing infrastructure (Sanaei *et al.*, 2014), to smaller infrastructure as part of an installation of mobile connection infrastructure (Satyanarayanan *et al.*, 2009), to local ad-hoc mini-clouds made up of other nearby hardware that would otherwise not traditionally be considered cloud computing hardware (Huerta-Canepa and Lee, 2010). The advantage of using dedicated cloud computing infrastructure is its availability and reliability, while that availability and reliability come at the increased cost of access as a utility, with billing for that access presenting additional potential complications (Sanaei *et al.*, 2014). Utilizing smaller cloud resources integrated with mobile connection infrastructure maintains some of those advantages while reducing latency, by placing the remote cloud servers closer to the mobile devices. Ad-hoc mini-clouds that utilize hardware that would otherwise be dormant and is already owned by end users provides offloading capabilities at minimal cost, but with less reliability, availability and less performance capability.

Numerous examples of commercialized applications that utilize MCC exist. There are generally two categories of these applications on the market. The first, and the simplest, contains those applications that provide access to much greater data storage resources, such as Dropbox or Microsoft's OneDrive, than would normally be feasible in a mobile device. While not strictly mobile-specific cloud computing implementations, these applications provide applications to ensure the synchronized access to a user's data from nearly any Internet-connected device the user might choose to use, and often those mobile applications provide mobile-specific features such as automatic photo backup.

The second category of applications is those that rely on cloud resources primarily for processing. Voice recognition applications such as Apple's Siri push the computational work off of the mobile device and onto a cloud server to reap a number of advantages (Johnson, 2013). However, in offloading to the cloud, these applications and others like them allow mobile devices to perform tasks which would otherwise place too large a burden on the processing resources of the mobile device to complete in a reasonable amount of time. This offloading of computation from mobile devices can also have an additional effect in reducing the energy consumption of a mobile device (Zhang *et al.*, 2015). In fact, this can be done even for jobs that would otherwise be feasible on a mobile device (Ma *et al.*, 2013).

1.1.1 MCC for energy saving

These gains are offset by increased energy consumption incurred during the transmission and reception of data required to support the use of these cloud resources. As energy consumption is intimately connected to battery life, the cost-benefit trade-off associated with utilizing cloud computing resources is of key importance.

The obvious scenario for examining this trade-off is that in which computation is offloaded specifically for reduced mobile energy consumption. The goal in this case is to minimize overall mobile device energy usage; if the device can save energy by offloading while achieving its other goals, it should, and if it can't it should perform that processing locally.

The primary performance measure in mobile cloud computing problems is either the total energy consumed by a mobile device or a set of mobile devices, or the total power. The factors that make up this total energy consumption can be varied and complex, but are often generalized to:

- Power (or in some cases energy) consumed for local job processing
- Power/energy consumed in transmitting and receiving data
- Power/energy consumed while idle waiting for remote job processing

Kumar and Lu (2010) formalize this simple analysis:

$$P_c \left(\tfrac{C}{M}\right) - P_i \left(\tfrac{C}{S}\right) - P_{tr} \left(\tfrac{D}{B}\right)$$

where P_c, P_i and P_{tr} are the power components detailed above, C is the amount of computation performed, M is the rate (in instructions per second) at which the mobile device computes, S is the same for the cloud system, D is the amount of data that must be exchanged to perform offloading, and B is the network bandwidth available. If the net result of this equation is positive, offloading should occur; this is essentially a simple mathematical expression of the trade-off described above.

Additionally, deadlines and execution time must enter the picture at some point - often the most energy-efficient computing schedule is one that performs the least amount of work as slowly as possible, but this is not a practical scheduling method for obvious reasons. Execution time is often defined as the total amount of time (in milliseconds or seconds) required to execute a task with the chosen combination of execution resources, transmission resources, and power-consumption settings. Execution time is made up of equivalent factors to the power/energy consumption factors above. These can also be subject to deadline constraints. Deadlines in cases where mobile devices are locally executing tasks are sometimes ignored, especially in cases where the focus is solely on scheduling remote tasks, such that only the remote/cloud execution case is subject to a deadline constraint.

One of the primary conclusions of this analysis is that offloading is not always power-efficient. While some applications are extremely computationally intensive and perform those computations on relatively small amounts of data, certain applications can require significant amounts of energy to successfully offload their data to the cloud, a requirement often driven by large amounts of data relative to the computational load generated by the application. In those cases, it can be very difficult to justify computational offloading from an energy-efficiency perspective. Namboodiri and Ghose (2012) demonstrate that in an example like the decoding of multimedia files, the large amounts of data render the offloading process detrimental. This is one way where an application that would be infeasible to process on a mobile device could remain infeasible despite the existence of MCC.

1.1.2 Challenges

Despite the fact that mobile cloud computing is generally based on existing cloud computing services (Sanaei *et al.*, 2014), there are differences from non-mobile-specific cloud computing applications, or even from existing uses of distributed computing. Traditionally, optimizations of distributed computing have focused on throughput in resource-constrained environments (Li *et al.*, 2009), and resources are usually expended in a cost-efficient fashion (Khan *et al.*, 2014). In MCC, the resources that are constrained do not necessarily affect computational throughput, but instead involve the availability and reliability of network connectivity, and the costs and energy consumption related to its usage as detailed above.

One challenge in effective implementation of mobile cloud computing is the heterogeneity of mobile architectures and operating systems. Even between Android and iOS, two popular mobile operating systems, there is little to no direct code reusability; both platforms utilize different programming languages, APIs, and SDKs. Despite ARM being generally accepted as the mobile architecture of choice, its use in server applications - though possible - is limited due to the dominance of the x86 instruction set, meaning that even MCC applications that target one specific platform could encounter difficulties. However, many of these issues are at least potentially avoidable with the use of virtualization, to allow cloud servers to run code not directly built for their architectures or operating systems; indeed, virtualization is considered a key feature in cloud computing (Ma *et al.*, 2013). Even still, this adds complexity and overhead, which is only avoidable with specifically-targeted applications. To take that route presents the trade-off of narrowing the usability of MCC and increasing development complexity.

Another challenge unique to mobile cloud computing is the network connection. While these assumptions do not hold true everywhere, it is usually safe to assume in traditional cloud computing applications that an Internet connection is generally available, fast, stable, and effectively cost-free (at least in terms of direct costs incurred for network connectivity for the application). However, none of these assumptions can be made about a mobile device using a wireless Internet connection. In many cases, cellular data is significantly more expensive than a traditional fixed-wire connection, and much of that cost comes from bandwidth metering which severely limits the amount of data that a user can transfer affordably. Even with sufficient connectivity, wireless connections often suffer lower performance - and less reliable performance - than fixed-wire connections (Aguayo *et al.*, 2004). On top of that, a wireless connection can drop out and reappear frequently and without warning, especially in areas with poor signal reception. And even a good wireless signal can introduce additional connection latency compared to a fixed line. All of these difficulties add up to scenarios where mobile devices can have a harder time offloading, with fewer opportunities and less success, than traditional cloud computing applications. These difficulties must be accounted for when evaluating mobile-specific cloud computing applications.

On top of these concerns, the existing concerns of security and privacy that are inherent in cloud computing continue to apply to MCC (Kumar and Lu, 2010). Private data may be required to leave a user's device and be transmitted across the Internet to the remote resources being used. To protect that data, encryption may be applied, but that places an additional computational burden on the mobile device, one that is impossible to offload. It also requires the end user to ultimately trust the service being implemented to handle the offloaded computation with their data, as even if the mobile device encrypts it before transmission, it often needs to be decrypted by the cloud server in order for it to be used. With that said, the broad adoption of cloud storage services suggests that most are comfortable with the use of these services from a privacy perspective for the time being.

1.1.3 Application models

A number of practical implementations of MCC have been developed for research purposes, in order to demonstrate its viability and explore its strengths and limitations. Khan *et al.* (2014) survey and categorize some of those implementations, some of which are detailed below.

One way to examine the implementations that exist is through their primary objective, whether it is to increase the performance available to mobile applications, to reduce the energy consumption required for those applications, to maximize the use of constrained resources, or to achieve a combination of these goals and perhaps more. A few of these different model categories are visible in the selected examples that follow. Several other models have been discussed and examined in various works, however the primary theme of all of these models is that of computational offload for the benefit of mobile devices.

Cuckoo (Kemp *et al.*, 2010) provides an offloading framework specifically for Android smartphones. This framework requires application developers to program specifically for the Cuckoo framework, developing the remote implementations that will replace local implementations when they are offloaded. While the authors lay the groundwork for intelligent offloading, they do not do anything more sophisticated in this paper than preferring remote execution when it is available; the model as it stands is theoretically capable of prioritizing multiple objectives, but the implementation as described is limited to a performance-based objective only. Cuckoo is flexible enough to be run on dedicated cloud infrastructure, cloudlets, or local ad-hoc clouds.

MAUI (Cuervo *et al.*, 2010) solves the problem of burdening the programmer with implementing offloading capabilities by automating the offloading and remote execution processes; all a programmer needs to do is suggest to the implementation what code sections are suitable for offloading. The limitation of MAUI's approach is that it is best served by managed code environments that are already inclined to support automatic cross-architecture execution with just-in-time (JIT) compilation.

It does, however, make a point of determining which code is genuinely worth offloading from both an energy and performance standpoint, making it an excellent example of a multi-objective-based model for MCC, applied for energy savings as well as for augmenting mobile resources.

ThinkAir (Kosta *et al.*, 2012), also a multi-objective model, extends MAUI's model to include offloading jobs from multiple devices. The other models described in this section focus solely on offloading from a given mobile device's perspective, where resource allocation is entirely out of the device's control. ThinkAir adds system-level control to manage server resources dynamically based on overall system load, in addition to making intelligent offloading decisions based on data collected during execution on the local devices.

CloneCloud (Chun *et al.*, 2011) takes MAUI's approach further in a different direction, by removing the need for any programmer input into offloading decisions whatsoever. It achieves this by implementing application-level virtual machines (VMs) which clone the local device state for offloading purposes. The disadvantage to this model is that it requires application processing to cease on the mobile device at time of offload, only to resume on completion of the offloaded computation and receipt of the results; this may be a tricky model to reconcile with the UI design principle of application responsiveness.

1.2 Literature review

1.2.1 Mobile cloud computing

The review on literature related to the issues investigated with this thesis begins with work relating to mobile cloud computing. Introductions to MCC are provided by Kumar and Lu (2010) and Ma *et al.* (2013). A number of in-depth surveys of the work done in this field have been conducted by Dinh *et al.* (2013), Khan *et al.* (2014), Fernando *et al.* (2013), Rahimi *et al.* (2013), and others. Meanwhile, Sanaei *et al.* (2014) present an evaluation and taxonomy of the heterogeneity of various aspects of MCC and how those affect and distinguish MCC from traditional cloud computing. Khan *et al.* (2013) evaluates the current state of security and privacy for MCC and further describes those challenges.

Early work on computational offloading from mobile devices was performed by Rudenko *et al.* (1998, 1999), with the first paper demonstrating the usefulness of the concept and the second laying out the logistical challenges of a scheme with a proposed framework for navigating them. Rong and Pedram (2003) furthers this work by coupling the offloading decision-making process with information about the mobile device's active power state management, and Lin *et al.* (2015) produces further work on this front with an effort towards energy reduction within minimal-delay schedules.

1.2.2 MCC and energy efficiency

Energy efficiency through MCC is primarily driven by the offloading of computation, as is described above in Section 1.1.1. In that section, the model for calculating the efficacy of that offloading proposed by Kumar and Lu (2010) is detailed. In that vein, Xian *et al.* (2007) describe a method for determining if offloading is worthwhile based on a break-even measure of computational effort, simplifying the determination of computational effort required by an application. Song *et al.* (2014) derive a more detailed model than that of Kumar and Lu describing that tradeoff between the energy consumed by offloading and by computation, and describe a method for using that information to make offloading decisions accordingly. As well, Zhang *et al.* (2013b; 2015) present several papers specifically pursuing optimal energy efficiency in MCC, the first given here defining the conditions under which offloading should occur within their system model, and the second going further into making energy-optimal offloading decisions.

It is common for a mobile device to have access to a variety of wireless networks. Klein *et al.* (2010) propose a means by which a mobile device can select between multiple available connections, using whatever measures of the quality of each available connection are relevant. Xu and Mao (2013) describe a similar energy model to Zhang *et al.* while investigating the differences between different wireless technologies. Barbera *et al.* (2013) study real-world applications for their usefulness in computational offloading, and the effects of using different wireless technologies on energy efficiency, while Lei *et al.* (2013) additionally investigates using MCC on combined heterogeneous networks.

1.2.3 Computation offloading

As described above, (Kemp *et al.*, 2010), (Cuervo *et al.*, 2010), (Kosta *et al.*, 2012), (Chun *et al.*, 2011), and many others provide, implement and evaluate various application models for computational offloading specifically in an MCC setting. Many of these models work towards improving various aspects of the practical details of implementing an MCC offloading system, including the logistics of implementing an offloading system, handling the appropriate balance between offloading and local processing, and managing jobs from multiple mobile devices, although they rarely account for all of the obstacles which impede the implementation of a complete and market-ready MCC system.

It is also discussed in the survey by Guan *et al.* (2011), who cover, among other things, some of the existing arguments and methods for computational offloading and dynamic partitioning. Valery *et al.* (2015) investigate the benefits of this partial offloading with adaptive partitioning of jobs, focusing on increasing utilization and reducing data transfer, the combination of which work towards energy efficiency as previously discussed. Partial offloading is also investigated by Yang *et al.* (2013) with a view towards maximizing throughput.

Yue *et al.* (2014) presents a model for partial offloading within a multiple-user environment sharing a constrained cloud server, in particular where some devices suffer from poorer channel conditions than others. As well, Yue (2015) further refines the approach used to encourage energy fairness, although with only the binary offloading case. This thesis differs from this work by integrating the partitioned offloading model with a system that has a single constrained, shared channel in addition to the constrained, shared server. It also defines and implements a distinctly different approach to improving energy fairness.

1.2.4 Job scheduling

In shared computing environments, it is likely that multiple jobs from multiple users will arrive and expect to use the shared resource(s) simultaneously; in these situations methods for scheduling those jobs are required in order to make the necessary decisions. This is the case for MCC as well, as demonstrated by Wan *et al.* (2015) who provide both a justification and a method for using shared, constrained server resources in support of mobile devices and MCC applications. The general efficacy of basic scheduling methods are described in several articles, with a form of FGFS being evaluated in Schwiegeishohn and Yahyapour (1998) and EDF being evaluated in Kruk *et al.* (2004). Indeed, scheduling problems are an old and well-covered topic, with numerous procedures detailed for optimizing schedules (one example is Adams *et al.* (1988)). However, the combination of tandem queues, local processing, and energy fairness optimization, as used in the system model described in Chapter 2, cause difficulties when attempting to fit this problem into an existing job shop scheduling model.

Of significance is the work by Barbarossa *et al.* (2013), which investigates jointly optimizing transmission and computation resources for minimizing energy consumption. Their work differs from this thesis first by focusing primarily on modeling channel conditions and handling fading and variability there, while also focusing on minimizing overall energy across all users as opposed to encouraging fairness across users.

Prior to their work on energy modeling and energy optimality, Zhang *et al.* (2013a) present a scheduling policy for energy-efficient collaborative execution which forms a part of that energy optimality work. As well, Li *et al.* (2014) describe a method for managing job schedules in environments with unstable connections which may cause offloading to fail.

Work has also been done regarding scheduling for MCC outside of energy efficiency. Achary *et al.* (2015) provide an ant colony optimization-based model for dynamically scheduling mobile cloud computing jobs for increased performance and job throughput. Liu *et al.* (2009) suggest a genetic algorithm-based scheduling model for MCC within a broader solution involving abstraction of heterogeneous distributed systems.

Although not strictly a scheduling technique, Wang and Dey (2013) present an interesting method for adaptively varying the job to be offloaded in an effort to mitigate varying network conditions.

1.3 Thesis overview and organization

The approach taken in this thesis involves a common scheduler which attempts to find a globally-optimum energy consumption across all considered mobile devices. It does so by offloading tasks for which the maximum benefit is derived from offloading, when not all otherwise-eligible tasks can be offloaded due to resource constraints. This maximum benefit is characterized by an interest in energy fairness, i.e., the attempt to ensure all devices have equal success in offloading data despite differences in opportunities caused by poor data rates or higher energy consumption in transmission.

The system model used throughout the remainder of the thesis defines the parameters and variables relevant to this scenario. This model is explored in Chapter 2. This system involves a shared channel used to upload jobs to a shared server, creating two constrained resources in tandem that need to be shared between mobile devices attempting to offload work to the cloud. These shared resources are divided amongst users in discrete time slots assigned to specific jobs. As well, an integer linear program (ILP) is defined to produce an optimal min-max offline schedule for a given input function of jobs and arrival times. This optimization problem attempts to minimize the largest device energy; a formulation is given for the joint channel and server scheduling scenario and its complexity is analyzed. This formulation is used to produce results which provide a lower bound and an example of optimal scheduling; these results are used to compare the performance of the other scheduling methods outlined in this thesis to that of the offline optimal schedule.

In Chapter 3, online scheduling methods are defined. In contrast to the offline optimal scheduler, the schedulers outlined in Chapter 3 are online schedulers with no information regarding future arrivals. In particular, the Average Queue Energy Prioritization (AQE) scheduling algorithm is defined, introduced, and its complexity analyzed. This scheduler makes scheduling decisions based on the information available to it from the jobs currently in the queue. Several other more basic methods are also detailed to present additional points of comparison for AQE.

With those schedulers defined, Chapter 4 provides results generated from simulations of those schedulers in order to evaluate and analyze them. Numerous aspects of the performance of these schedulers, including average and minimum/maximum energy performance, are compared. As well, the parameters relevant to AQE's performance are defined and a method for determining their optimal values proposed.

Finally, Chapter 5 presents the conclusions of this thesis, and suggests potential future research derived from this work.

Chapter 2

System Model and Problem Formulation

2.1 Overview

In this chapter, the system model used throughout this thesis is introduced. This model describes the assumptions made about the mobile devices, channel, channel conditions, and server parameters which form the environment in which the scheduling methods described in Chapters 2 and 3 are evaluated. This model is derived from that of Yue (2015), extended to cover the joint channel/server scheduling conditions evaluated in this thesis, and generalized to a form of non-binary partial offloading in a similar manner as Yue *et al.* (2014).

As well, a problem is formulated to provide optimum scheduling in a non-causal, or offline, environment, for the system model described. This model is investigated in detail to demonstrate its relevance to the system, and to the online algorithms to be detailed in a later chapter.

2.2 System model

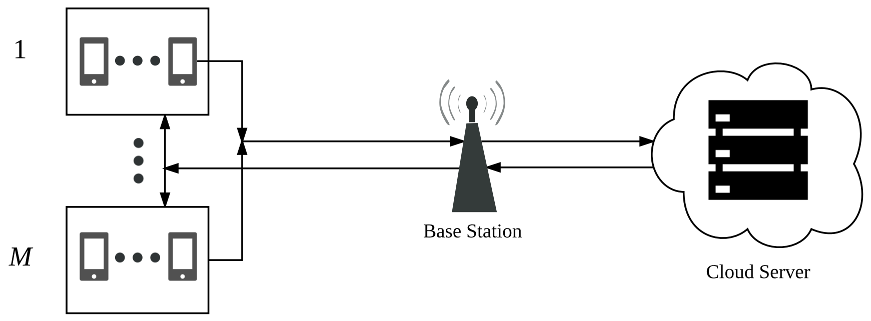

Figure 2.1: Diagram representing the system model

To start we assume a set of mobile device classes $\mathcal{M}$, with each class $m \in \{1...M\}$. The mobile devices within a given mobile device class m generate a set of jobs to be processed, $\mathcal{J}_m$, containing jobs that can either be offloaded, in full or in part, or processed locally, with each job $j \in \{1...J\}$.

The processing of these jobs is modeled in discrete time, with each time slot $t \in \{1...T\}$ contained in the set of all time slots $\mathcal{T}$. In addition, each job can be divided into partitions, in which one portion of the job is offloaded and the remainder is processed locally. The chosen partition is defined as $p_{m,j} \in \{1...P_{m,j}\}$, i.e., there are $P_{m,j}$ ways in which a job can be divided between local and remote execution. The possible partitions are in the set $\mathcal{P}_{m,j}$. A special case $p_{m,j} = 1$ is defined as local-only execution, with no remote offloading.

When these jobs are fully processed locally, they require $l_{m,j,1}$ units of processing time on the local device. For offloading a job (m, j) at a given partition setting p, $u_{m,j,p}$ time slots are required to upload required data on the channel, and $r_{m,j,p}$ time slots are required to process that data at the server. Without loss of generality we order partitions such that larger partitions provide more offloading, i.e., $r_{m,j,p_1} > r_{m,j,p_2}$ if $p_1 > p_2$. The rate of uploading and remote processing available to each device is given as U_m and R_m processing units per time slot, respectively. This results in $l_{m,j,1} - r_{m,j,p}\ R_m$ processing units being required to process the local portion of the selected partition locally. The energy involved in receiving data wirelessly is assumed to be much lower than that of transmission (Miettinen and Nurminen, 2010), and so can be ignored for the purposes of this model (Zhang *et al.*, 2013b).

It is assumed that at least one device class experiences a disadvantage relative to the other classes, which is a decreased transmission rate per channel, or to say it another way, an increased number of channel time slots required to complete offloading to the server. When discussing this disadvantage, parameters relating to advantaged device classes (i.e., those with good channel conditions) are subscripted accordingly. For example, in the case of two classes - one disadvantaged and one advantaged - parameters relating to the advantaged class are subscripted with G, and those relating to the disadvantaged device class with bad channel conditions are subscripted with B. For example, it is common to refer to the relative arrival rate of disadvantaged to advantaged users as $\lambda_B/(\lambda_B + \lambda_G)$. From here, if multiple bad or good device classes exist, they can be indexed, e.g. $B_1, B_2, ...B_n$. In general, channel rate differences between the classes are expressed as a ratio, such as U_B/U_G.

All device classes are assumed to share the same channel, which is divided into a set of $\mathcal{Z}$ channel time slots. These time slots are used by the scheduler as follows:

$$z_{m,j,t} = \begin{cases} 0 & \text{if channel time slot } t \text{ is not assigned to mobile/job } (m,j) \\ 1 & \text{if channel time slot } t \text{ is assigned to mobile/job } (m,j) \end{cases} \tag{2.1}$$

As well, all device classes are assumed to share the same server, which is divided into a set of $\mathcal{Y}$ server time slots. These time slots are used by the scheduler as follows:

$$y_{m,j,t} = \begin{cases} 0 & \text{if server time slot } t \text{ is not assigned to mobile/job } (m,j) \\ 1 & \text{if server time slot } t \text{ is assigned to mobile/job } (m,j) \end{cases} \tag{2.2}$$

Finally, the scheduler must choose how much of a job is to be offloaded as part of the scheduling decision. In order to account for partitioning decisions made, $x_{m,j,p}$ is used to indicate whether or not a given partition $p_{m,j}$ is selected for a given mobile device class and job:

$$x_{m,j,p} = \begin{cases} 0 & \text{if partition } p \text{ is not selected for mobile/job } (m,j) \\ 1 & \text{if partition } p \text{ is selected for mobile/job } (m,j) \end{cases} \tag{2.3}$$

Each job arrives at the channel at time slot $a_{m,j}$; it is also assumed throughout this thesis to have been generated by the mobile device at this time, although that is not necessarily the case. The job must be completed in its entirety - including local processing, transmission, and server processing - before a deadline $D_{m,j}$. (Note that $D_{m,j}$ is not subscripted with p, i.e., it is not dependent on the selected partition.) Therefore the scheduler must select $u_{m,j,p}$ channel time slots and $r_{m,j,p}$ server time slots for the job, for the given selected partition $p_{m,j}$, and for values of $t \in \mathcal{T}_{m,j}$, where $\mathcal{T}_{m,j} = \{a_{m,j}, ..., D_{m,j}\}$. If there are not enough free time slots, the scheduler must select a different value of $p_{m,j}$ for the given job (m,j) by setting the appropriate values for $x_{m,j,p}$, or otherwise reassign the time slots that are not free.

Energy costs and fairness

Each unit of local processing, and each time slot assigned to a device for offloading on the shared channel, incur an energy cost to the given mobile device. Q^l_m is defined as the energy required per time unit for local processing; therefore the energy required for local processing of a given job (m, j) and selected partition $p_{m,j}$ is defined as $\mathcal{E}^l_{m,j,p} = Q^l_m \, l_{m,j,p}$. Likewise, Q^u_m is defined as the energy required for mobile device m to transmit on one time slot, which gives the energy required for offloading of a given job (m, j) for the selected partition $p_{m,j}$ as $\mathcal{E}^u_{m,j,p} = Q^u_m \, u_{m,j,p}$. From there, the total energy consumed by each device for each job, given as $\mathcal{E}_{m,j}$, is defined as:

$$\mathcal{E}_{m,j} = \sum_p x_{m,j,p} \, \mathcal{E}_{m,j,p} = \sum_p x_{m,j,p} \, (\mathcal{E}^l_{m,j,p} + \mathcal{E}^u_{m,j,p}) \tag{2.4}$$

assuming $\sum_p x_{m,j,p} = 1$, i.e., only one partition is selected.

Much of this thesis is focused on the concept of energy fairness. This is defined for the purposes of this thesis as the minimization of the difference in average energy consumption between two device classes. The average energy consumption for a given class m can be defined as follows:

$$\overline{\mathcal{E}_m} = \frac{\sum_{j \in \mathcal{J}_m} \mathcal{E}_{m,j}}{J_m} \tag{2.5}$$

With that, the difference in average energy consumption between, for example, classes 1 and 2, can be defined as:

$$\Delta_{\mathcal{E}} = |\overline{\mathcal{E}_1} - \overline{\mathcal{E}_2}| \tag{2.6}$$

From there, in general the objective when attempting to achieve energy fairness is to minimize $\Delta_{\mathcal{E}}$.

In addition to this fairness objective, the schedulers used with this model are intended to work towards an optimization objective:

$$\min \max_{m} \sum_{j \in \mathcal{J}_m} \mathcal{E}_{m,j} \tag{2.7}$$

that is, the minimization of the maximum-energy device class, in order to achieve fairness (or any other objective) in a way that minimizes the overall energy consumed by all devices.

2.3 ILP formulation

This model is first used below to formulate an integer linear program (ILP) for finding an optimal schedule, to be used as a bound on the performance of the online schedulers discussed in Chapter 3. The formulation given below is derived from a similar problem, in which only the server is constrained (Yue *et al.*, 2014). It is extended here to add the constraints relating to the shared channel, while making a few simplifications. As in the original formulation, this ILP is given all inputs to the system at the start, in an offline fashion, and allowed to derive an optimal energy schedule from that complete information. The formulation is first defined, and then discussed.

$$\text{minimize } \max_{m} \sum_{m\in\mathcal{M}} \sum_{j\in\mathcal{J}_m} \sum_{p\in\mathcal{P}_{m,j}} x_{m,j,p}\, \mathcal{E}_{m,j,p} \tag{2.8}$$

subject to

$$\sum_{p\in\mathcal{P}_{m,j}} x_{m,j,p} = 1, \forall\, m \in \mathcal{M}, j \in \mathcal{J}_m \tag{2.9}$$

$$\sum_{m\in\mathcal{M}} \sum_{j\in\mathcal{J}_m} y_{m,j,t} \leq 1, \forall\, t \in \mathcal{T} \tag{2.10}$$

$$\sum_{m\in\mathcal{M}} \sum_{j\in\mathcal{J}_m} z_{m,j,t} \leq 1, \forall\, t \in \mathcal{T} \tag{2.11}$$

$$\sum_{t\in\mathcal{T}_{m,j}} y_{m,j,t} \geq x_{m,j,p}\, r_{m,j,p}, \forall\, m \in \mathcal{M}, j \in \mathcal{J}_m, p \in \mathcal{P}_{m,j} \tag{2.12}$$

$$\sum_{t\in\mathcal{T}_{m,j}} z_{m,j,t} \geq u_{m,j,p}\, x_{m,j,p}, \forall\, m \in \mathcal{M}, j \in \mathcal{J}_m, p \in \mathcal{P}_{m,j} \tag{2.13}$$

$$1 - y_{m,j,t} \geq \max_{\tau} z_{m,j,\tau}, \forall\, m \in \mathcal{M}, j \in \mathcal{J}_m, t \in \mathcal{T}, t \leq \tau \leq T \tag{2.14}$$

$$x_{m,j,p} \in \{0,1\}, \forall\, m \in \mathcal{M}, j \in \mathcal{J}_m, p \in \mathcal{P}_{m,j} \tag{2.15}$$

$$y_{m,j,t} \in \{0,1\}, \forall\, m \in \mathcal{M}, j \in \mathcal{J}_m, t \in \mathcal{T} \tag{2.16}$$

$$z_{m,j,t} \in \{0,1\}, \forall\, m \in \mathcal{M}, j \in \mathcal{J}_m, t \in \mathcal{T} \tag{2.17}$$

This formulation minimizes the largest device class energy out of all available device classes, in effect finding the minimum fair energy for all devices. A number of the constraints are from the original implementation from which this was derived, while constraints for the channel are related to the server constraints.

Constraint 2.9 ensures only one of the possible partition selections is made, while constraints 2.10 and 2.11 ensure that each of the time slot variables for channel and server time slots are only assigned to one job at a time. Constraints 2.12 and 2.13

ensure that at least the required number of time slots for uploading and processing are selected, for the given partition. It should be noted that constraints referring to non-preemptive scheduling have been removed, as this system model is preemptive.

One of the most significant constraints added to this formulation is constraint 2.14, which ensures that the entire portion of data to be offloaded (as decided by the selected partition) is uploaded to the server before server processing is allowed to begin. It does so by effectively taking the sign of the sum of all assigned channel slots forward of this time slot, since each value of $z_{m,j}$ is binary, and forcing sever time slots to be zero until no future channel time slots are assigned. The impact of this constraint is discussed in section 2.3.1.

Finally, constraints 2.15 through 2.17 provide binary selection variables for the offloading partition selection for each job, as well as for the selection of channel and server time slots for each job.

In producing the optimal offline min-max energy, this scheduler acts as a lower bound on the online schedulers as far as the maximum-energy user class is concerned. It does not, however, explicitly produce a lower bound on any class other than that with the highest energy consumption in a given schedule, and it only produces a lower bound for algorithms focused on energy fairness as opposed to average energy; this should be kept in mind while reviewing the results in Chapter 4.

2.3.1 Complexity

Yue (2015) presents a proof that a formulation of a similar problem, related to the one on which this optimum min-max optimization problem is based, is NP-complete. As the formulation set out in that paper is merely a special case of the more general formulation derived in this chapter, by extension this problem is also NP-complete.

NP-completeness does not necessarily mean intractability. However, one of the subtle complexities in solving this ILP comes from constraint 2.14. (For clarity this will be referred to as a *constraint definition*, from which all of the actual constraints for a given input are derived.) While it is not immediately apparent, this equation defines one new constraint for not just every time slot in a given system, but for every channel time slot decision for every job and user.

The number of individual explicit constraints derived from this constraint definition grows roughly with the square of the number of jobs. This can be seen in the derivation shown in equations 2.18 through 2.21, letting C be the number of explicit constraints generated for the solver by constraint 2.14, and letting $\overline{D}$ represent the average of $D_{m,j}\ \forall\ (m, j)$, i.e., the average of all job deadlines.

$$C = MT \max J_m \tag{2.18}$$

$$T \approx M\overline{D} \max J_m \tag{2.19}$$

$$C \approx (\max J_m)^2\, M^2\overline{D} \tag{2.20}$$

$$\therefore\ C \in \mathcal{O}(n^2) \tag{2.21}$$

This constraint definition is actually an improvement on the naive solution to constraining server processing to only occur after upload requirements are completely fulfilled:

$$y_{m,j,\tau} \leq 2 - z_{m,j,t} - x_{m,j,p}, \forall m \in \mathcal{M}, j \in \mathcal{J}_m, p \in \mathcal{P}_{m,j}, \tau \leq t \tag{2.22}$$

This formulation of the constraint definition forces one new constraint for every time slot prior to the current time t, which would allow the number of constraints required to schedule a given input to grow with the square of the number of *time slots*.

Especially when deadlines and job sizes are set as they are for most of the results in Chapter 4, the number of constraints grows rapidly, even with this improvement. The issue of this growth in the number of constraints is compounded as this is an ILP, and additional constraints rapidly increase the amount of time required for a branch-and-bound solver to identify the optimum solution. This difficulty is the justification for the simplifications made when solving this optimization problem for Chapter 4.

2.4 Summary

In this chapter, the parameters relevant to the system model used in this thesis have now been defined. These parameters will be referred to throughout the remainder of the thesis to define the scheduling methods outlined in Chapter 3, and to describe and interpret the results demonstrated in Chapter 4. As well, the offline scheduling min-max optimization problem was introduced and detailed. Some of the challenges inherent in solving this optimization problem for the system model described were also analyzed. Despite these challenges, this formulation will be used to provide a lower bound to the online scheduling algorithms described in Chapter 3 when the performance of those schedulers is discussed in Chapter 4.

Chapter 3

Online Scheduling

3.1 Overview

In this section, several online scheduling algorithms are introduced. In contrast to the offline scheduling optimization problem previously set out in Chapter 2, these schedulers do not receive any information about future job arrivals; they are required to evaluate new jobs causally, as they arrive, in the context of previous arrivals only.

Prior to a discussion on scheduling algorithms, an algorithm is defined to determine offloading feasibility. Then, three scheduling methods are introduced: First Generated First Served (FGFS), Earliest Deadline First (EDF), and Average Queue Energy Prioritization (AQE). These algorithms are described in detail, including an analysis of their complexity.

3.2 Determining queue state validity

Before discussing scheduling algorithms, it is necessary to describe the method in which a cloud server identifies whether or not a given job can be offloaded. This seems a trivial task on its face - examine all jobs in the queue ahead of the current job, and if they complete before this job needs to finish being served, then this job can be offloaded. The calculation is complicated, however, by the coupling of the shared channel and server queues; a job can reach the server and begin being processed, but then be pre-empted by a prior job which is earlier in the queue than the current job on the server but required more time to offload, or arrived later, etc. In addition, it may be that the channel and server queues are sorted differently. The completion time of each job is dependent on the job ahead of it in each of the queues, meaning that for any given job and any perturbation of the queues, all channel and server completion times need to be carefully re-calculated, in the correct order.

It may also seem plausible to stop searching as soon as a job is reached in a queue for which there are not enough time slots available to it before its deadline expires, i.e. based solely on the number of remaining unassigned time slots. However, one of the features of the AQE algorithm depends on the identification of jobs suitable for removal from the queue, and so this algorithm identifies completion times for all jobs that are waiting, and adds any that are no longer feasible given the current queue state to $\mathcal{F}$ for further examination if necessary.

The process used by these algorithms is to determine the validity of the queue as a whole after the insertion of a new job in the intended location. The scheduling algorithm then decides what to do about an invalid queue state. To do this, we begin by letting $\mathcal{Q}^c$ represent the set of all jobs accepted by the cloud server waiting for

Algorithm 1 Queue state validity

```
1: Sort $\mathcal{Q}^c$
2: for each job $q_n^c$ in $\mathcal{Q}^c$: do
3:    $m, j$ are from the job designated by $q_n^c$
4:    $S^c = a_{m,j}$
5:    starting at $t = S^c$:
6:    while $x_{m,j,p}u_{m,j,p} > \sum_t z_{m,j,t}$ do
7:       if $\sum_{\forall m,j} z_{m,j,t} = 0$ then
8:          $z_{m,j,t} = 1$
9:       end if
10:      $t = t + 1$
11:   end while
12:   Channel completion time $C_n^c$ = the final value of $t$
13: end for
14: Temporarily add $\mathcal{Q}^c$ to $\mathcal{Q}^s$
15: Sort $\mathcal{Q}^s$
16: for each job $q_n^s$ in $\mathcal{Q}^s$: do
17:   $m, j$ are from the job designated by $q_n^s$
18:   $S^s = C_n^c$
19:   starting at $t = S^s$:
20:   while $x_{m,j,p}r_{m,j,p} > \sum_t y_{m,j,t}$ do
21:      if $\sum_{\forall m,j} y_{m,j,t} = 0$ then
22:         $y_{m,j,t} = 1$
23:      end if
24:      $t = t + 1$
25:   end while
26:   Server completion time $C_n^s$ = the final value of $t$
27: end for
28: for all job $n$ in $\mathcal{Q}^c$ or $\mathcal{Q}^S$ do
29:   if $C_n^s > D_{m,j}$ then
30:      add $n$ to $\mathcal{F}$
31:   end if
32: end for
33: if $\mathcal{F} = \emptyset$ then
34:   Queue state is valid
35: else
36:   Queue state is invalid
37: end if
38: Remove $\mathcal{Q}^c$ from $\mathcal{Q}^s$
39: Sort $\mathcal{Q}^s$
```

uploading, and $\mathcal{Q}^s$ the set of all jobs that have been uploaded and are waiting for processing (both include jobs partially uploaded or processed, and jobs currently being uploaded or processed). For each job q_n^c in the channel queue, its earliest available time slot S^c is determined based on its arrival time, as shown in line 4. It then is assigned all free time slots forward of that slot in lines 6 through 11; those slots are marked as assigned. The channel completion time C_n^c is stored in line 12 for use as the earliest server time slot S^s, in line 18. Then all jobs from $\mathcal{Q}^c$ are temporarily added to $\mathcal{Q}^s$ and the process is repeated there, as shown in lines 16 to 27.

Finally, at the end of this process, if any jobs violate a deadline constraint they are added to $\mathcal{F}$ for later examination; this process appears in lines 28 to 32. The queue state is determined in lines 33 to 37, simply by determining if the set of jobs with failed deadlines F is empty. The jobs waiting for channel transmission are then removed from Q^s.

The complexity of this feasibility test is a critical portion of the complexity of the algorithms that implement it. Fortunately, it is straightforward to determine. For each job waiting, regardless of where it waits, it will require $u_{m,j,p}$ time slots to be selected out of $\mathcal{T}_{m,j}$, followed by $r_{m,j,p}$ time slots. This is linear in the number of time slots available to each job; therefore this process is linear in the number of jobs waiting, or $\mathcal{O}(n)$.

3.3 First Generated First Served

This algorithm is a simple modification to the simple first-come, first-served method of scheduling, namely that queueing decisions need to be made before a job enters the queue, as opposed to at time of service (i.e. after uploading is complete). This is related to the FGFS algorithm defined by Yue (2015), but is modified to accommodate the shared channel in addition to the shared server. Upon job arrival, the new job is inserted into the queue where it is automatically sorted to the back, as it is the last job to arrive. The validity of the queue is determined using Algorithm 1. If the queue is invalid with the new job arrival, $p_{m,j}$ is reduced and the process repeated until either the queue state is valid, or $p_{m,j} = 0$ and the job is not offloaded.

Algorithm 2 FGFS

```
1:  A job $(m, j)$ is generated at time $t_{m,j}$.
2:  Mobile $m$ calculates $p^*_{m,j} = \arg\min_p \ \mathcal{E}_{m,j,p}$.
3:  Mobile $m$ sends an upload request to the cloud server.
4:  The cloud server inserts the job into $\mathcal{Q}^c$ and sorts it according to $a_{m,j}$
5:  The cloud server determines queue state using Algorithm 1
6:  if queue state is valid then
7:      job is accepted at the current value of $p_{m,j}$
8:  else
9:      while $p_{m,j} > 0$ and queue state is invalid do
10:         $p_{m,j} = p_{m,j} - 1$
11:         The cloud server determines queue state using Algorithm 1
12:     end while
13: end if
14: return $p_{m,j}$
```

FGFS evaluates the queue validity a maximum of $p^*_{m,j}$ times; therefore, like the queue validity function, FGFS is also linear in the number of jobs waiting, or $\mathcal{O}(n)$.

3.4 Earliest Deadline First

This algorithm is very similar to FGFS, however instead of sorting the arrival into the queue based on its arrival rate, or $a_{m,j}$, it is sorted based on its deadline, or $D_{m,j}$. It is included under the premise that jobs from disadvantaged users will have a more difficult time meeting their deadline constraints, and so for jobs with shorter constraints, they are prioritized and therefore hopefully have a greater opportunity to successfully offload.

Algorithm 3 EDF

```
1: A job $(m, j)$ is generated at time $t_{m,j}$. Mobile $m$ calculates $p^*_{m,j}$.
2: Mobile $m$ sends an upload request to the cloud server.
3: The cloud server inserts the job into $\mathcal{Q}^c$ and sorts it according to $D_{m,j}$
4: Execute lines 5 to 13 in Algorithm 2
5: return $p_{m,j}$
```

Like FGFS, EDF is linear in the number of jobs waiting, or $\mathcal{O}(n)$.

3.5 Average Queue Energy Prioritization

One of the major drawbacks to both FGFS and EDF scheduling is their lack of knowledge of the energy performance of the users being serviced. Both algorithms accept jobs primarily based on the feasibility of a new arrival based on the current jobs in the queue and the intended position of the new job in the queue. This has several inherent flaws:

1. Unless deadlines are the intended prioritization measure - which is not the case when attempting to achieve energy fairness - neither FGFS nor EDF provide a means to prioritize one job over another.

2. If a new job arrives and cannot be accepted at its intended location in the queue, it is rejected outright, regardless of whether or not another permutation of the queue would allow it to be offloaded feasibly.

3. If a new job arrives in the queue which the system would otherwise prefer over an existing job for any reason, there is no mechanism to remove a job from the queue in favour of the new arrival. While this can be solved pre-emptively by flow control methods, such as the γ-modulated algorithms derived by Yue (2015), those methods still have an opportunity to apply the wrong level of flow control, and have little recourse against that once committed.

AQE attempts to solve all of these problems by evaluating multiple queue positions at each arrival, allowing what will be referred to in this paper as *soft revocation* - the ability to remove jobs from the queue when it is feasible to do so - to correct previous decisions when new information is available, and to select the optimum prioritization of new jobs based on the information available at the time.

Algorithm 4 describes the AQE process in detail, in particular lines 7 to 27. First, for a given job (m, j), lines 2 to 4 first allow the base job insertion algorithm to insert the job into its initial place in the job queue. At line 7, the search through the useful offload partitions for valid queue states begins, while lines 9 to 25 search through the possible queue positions forward of the sorted queue position, incrementing the current job's queue position by manipulating its "arrival time" at line 24. The algorithm will continue to further prioritize the newly-arrived job, moving it further ahead in the job queue until moving farther forward will violate a deadline that cannot be resolved within that queue arrangement. Finally, it will remove the job from the queue and assume it will be processed entirely locally. This algorithm allows the maximum opportunity for a new job to be placed in the queue while avoiding an exhaustive search of all permutations of those jobs waiting for service.

Algorithm 4 AQE

```
1: A job $(m, j)$ is generated at time $t_{m,j}$.
2: Mobile $m$ calculates $p^*_{m,j} = \arg\min_p \mathcal{E}_{m,j,p}$.
3: Mobile $m$ sends an upload request to the cloud server.
4: The cloud server inserts the job into $\mathcal{Q}^c$ and sorts it according to $a_{m,j}$
5: if $|\mathcal{Q}^s| + |\mathcal{Q}^s| < C_q$ then
6:   $p_{m,j} = p^*_{m,j}$.
7:   while $p_{m,j} > 0$ do
8:     Define $q^c_n$ as the $n$th position in the channel queue $\mathcal{Q}^c$
9:     while $q^c_n \neq q^c_1$ do
10:      The cloud server determines queue state using Algorithm 1
11:      if queue state is valid then
12:        $\Delta_{\mathcal{E}} = |\sum \mathcal{E}_A - \sum \mathcal{E}_D|$
13:        add the current queue state, job settings, and associated $\Delta_{\mathcal{E}}$ to $\mathcal{V}$
14:      else
15:        if all jobs in $\mathcal{F}$ are in $\mathcal{F}_V$ then
16:          remove all jobs in $\mathcal{F}$ from $\mathcal{Q}^c$ and $\mathcal{Q}^s$
17:          $\Delta_{\mathcal{E}} = |\sum \mathcal{E}_A - \sum \mathcal{E}_D|$
18:          add the current queue state, job settings, and associated $\Delta_{\mathcal{E}}$ to $\mathcal{V}$
19:        else
20:          exit the loop and go to line 26
21:        end if
22:      end if
23:      let $a_{n-1}$ be the arrival rate of $q^c_{n-1}$, i.e. the next job ahead of this job in
         the queue
24:      $a_{m,j} = a_{n-1} - \delta$ where $\delta$ is a small value placing this job only just ahead
         of the next job in the queue
25:    end while
26:    $p_{m,j} = p_{m,j} - 1$
27:  end while
28: else
29:   Perform lines 5 to 13 from Algorithm 2 without modification.
30: end if
31: The selected queue state and $p_{m,j}$ are chosen from $\mathcal{V}$ according to $\min \Delta_{\mathcal{E}}$; for
    those states with equal $\Delta_{\mathcal{E}}$, the arrival rate closest to the original $a_{m,j}$ is selected
32: return $p_{m,j}$
```

The measure AQE uses to make its decisions is the difference in average job energy between the target user class (in the case of this paper, users who suffer additional energy costs in offloading jobs) and the most-advantaged user class, a value referred to here as the *energy delta*. The absolute value is taken to discourage over-prioritization of the target class of users. This value is used as follows.

Let $\mathcal{V}$ be the set of all valid, found permutations of the newest job, and $\mathcal{E}_A$ the set of job energies from all advantaged jobs and $\mathcal{E}_D$ that of disadvantaged users in the queue. Additionally, let $\mathcal{F}_V$ be the set of all jobs with failed deadlines that are removable from the queue under the soft revocation rules described below.

At each of the valid queue positions examined (including the fully-local "position") the energy delta is calculated, the queue state is captured and the value is stored in set $\mathcal{V}$; these steps are shown in lines 17 and 18. In line 31, at the conclusion of the above process, the queue position providing the minimal energy delta is chosen. For queue positions that share equal energy deltas, the position closest to the original insertion position is chosen, to again avoid over-prioritization or other poor performance.

A version of this algorithm is also possible using an EDF-style queue sorting approach instead of FGFS, as seen in Algorithm 5.

The AQE process addresses the prioritization and queue feasibility concerns set out at the beginning of this section. To address the inability to correct previous decisions based on new information, the aforementioned soft revocation method is implemented. For any queue position, previously-accepted jobs can be removed from the queue, although this is done if and only if the following restrictions are respected:

1. the newly-arrived job can meet its deadline constraint given its position in the current queue order;

2. the newly-arrived job is from the target (or disadvantaged) user class;
3. none of the jobs to be removed from the queue are from the target user class.
4. none of the jobs to be removed from the queue have started the offloading process; and
5. none of the jobs to be removed from the queue will fail their deadline constraint if processed locally starting at the time of removal from the queue.

These restrictions do require some justification. Item 1 is a fairly obvious restriction. Items 2 and 3 serve as a simple way to ensure the target user class is prioritized. However, it is items 4 and 5 that give this revocation process its “softness”; jobs are only removed from the list when removing them will not differ from never having been selected for offloading at all (outside of the increased cost of local processing, of course, which is arguably the point of removing them). In theory, jobs could be removed even if those restrictions were violated, but that would complicate the case for doing so and the argument to be made for that process is less clear.

Queue size threshold

One parameter of note in the given AQE algorithms is C_q, the queue size threshold, utilized in line 5. Below this threshold, the AQE algorithm is not utilized and the underlying algorithm (in this case either FGFS or EDF) is relied upon wholly. This threshold is utilized to prevent the AQE algorithm from manipulating newly-arrived jobs in the event that the queue does not contain a sufficient number of jobs on which to reliably base an energy-optimal decision.

The justification for this threshold comes from cases where there are two classes, one with disadvantaged users and one with advantaged users, and the queue size

Algorithm 5 AQE+EDF

```
1: A job $(m, j)$ is generated at time $t_{m,j}$. Mobile $m$ calculates $p^*_{m,j}$.
2: Mobile $m$ sends an upload request to the cloud server.
3: The cloud server inserts the job into $\mathcal{Q}^c$ and sorts it according to $D_{m,j}$
4: if $|\mathcal{Q}^s| + |\mathcal{Q}^s| < C_q$ then
5:     while $p_{m,j} > 0$ do
6:         while $q^c_n \neq q^c_1$ do
7:             The cloud server determines queue state using Algorithm 1
8:             if queue state is valid then
9:                 $\Delta_{\mathcal{E}} = |\sum \mathcal{E}_A - \sum \mathcal{E}_D|$
10:                add the current queue state, job settings, and associated $\Delta_{\mathcal{E}}$ to $\mathcal{V}$
11:            else
12:                if all jobs in $\mathcal{F}$ are in $\mathcal{F}_V$ then
13:                    remove all jobs in $\mathcal{F}$ from $\mathcal{Q}^c$ and $\mathcal{Q}^s$
14:                    $\Delta_{\mathcal{E}} = |\sum \mathcal{E}_A - \sum \mathcal{E}_D|$
15:                    add the current queue state, job settings, and associated $\Delta_{\mathcal{E}}$ to $\mathcal{V}$
16:                else
17:                    exit the loop and go to line 23
18:                end if
19:            end if
20:            let $D_{n-1}$ be the arrival rate of $q^c_{n-1}$, i.e. the next job ahead of this job in
               the queue
21:            $D_{m,j} = D_{n-1} - 1$
22:        end while
23:        $p_{m,j} = p_{m,j} - 1$
24:    end while
25: else
26:    Perform lines 5 to 13 from Algorithm 2 without modification.
27: end if
28: The selected queue state and $p_{m,j}$ are chosen from $\mathcal{V}$ according to $\min \Delta_{\mathcal{E}}$; for
    those states with equal $\Delta_{\mathcal{E}}$, the arrival rate closest to the original $D_{m,j}$ is selected
29: return $p_{m,j}$
```

is smaller than the ratio of bad arrivals, λ_B, to good arrivals, λ_G. In this event it may appear that no bad arrivals have managed to enter the queue, even though no bad arrivals have arrived recently enough to have even tried to enter the queue. By ensuring the queue has sufficient length to provide the AQE algorithm with a reasonable snapshot of the system, the algorithm performs significantly better. The selection of this parameter is examined in Chapter 4.

Complexity

The complexity of this algorithm is as follows: the queue state validity check, which is $\mathcal{O}(n)$, is executed up to $n\,p_{m,j}$ times. This algorithm is therefore quadratic in n (the number of jobs waiting in the queue), i.e. $\mathcal{O}(n^2)$.

3.6 Summary

In this chapter, a number of online scheduling options were introduced, in particular the Average Queue Energy Prioritization method for increasing energy fairness, as well as First Generated First Served and Earliest Deadline First for points of comparison. These methods will serve as comparison points when discussing energy fairness in Chapter 4.

Chapter 4

Performance Results

4.1 Overview

In this chapter, the performance of each online scheduler described in Chapter 3 is compared to the others, in both energy fairness and overall energy performance. As well, the performance of the schedulers for disadvantaged users is compared against the solution of a comparable offline schedule based on the formulation described in Chapter 2. These comparisons occur at a variety of parameters to demonstrate their efficacy in a range of situations, as opposed to directly targeting specific applications. In each comparison scenario, the relevant parameters used to generate those results are given, followed by the results.

It is important to note several assumptions held throughout these results. First, it is assumed these jobs arrive according to a Poisson arrival process, with arrival rate λ_m for the mth device class. As well, it is assumed that each partition $p_{m,j}$ represents processing a fraction of the units required for a fully-local or fully-offloaded job, with

$p_{m,j} = 1$ as the fully-local case and $p_{m,j} = P_{m,j}$ as the fully-offloaded case, as follows:

$$l_{m,j,p} = l_{m,j,1} \frac{P_{m,j} - p_{m,j}}{P_{m,j} - 1} \tag{4.1}$$

$$r_{m,j,p} = r_{m,j,P_{m,j}} \frac{p_{m,j} - 1}{P_{m,j} - 1} \tag{4.2}$$

$$u_{m,j,p} = u_{m,j,P_{m,j}} \frac{p_{m,j} - 1}{P_{m,j} - 1} \tag{4.3}$$

Additionally, instead of manipulating the number of user classes present in these results to vary the relative loads of advantaged and disadvantaged users, the relative arrival rates of two classes are varied, one disadvantaged and one advantaged in terms of the channel rate available to them. Accordingly, these results only contain two user classes. The arrival rates are given as $\lambda_B/(\lambda_B + \lambda_G)$, i.e., the fraction of the overall arrival rate attributable to disadvantaged users. This scenario is comparable to one with more than two user classes, still with one disadvantaged class and where all advantaged classes are homogeneous in their channel rates, in which case the denominator of the relative arrival rates is effectively equal to M. This change is made to improve the clarity of the discussion of these results.

4.2 Algorithm performance

4.2.1 Channel-constrained results

These results demonstrate the feasibility of the online scheduling methods by comparing them to a solution of a comparable offline schedule, acting as a lower bound on energy. This was generated by solving the formulation described in Chapter 2. These results were generated with the parameters set out in Table 4.1.

Simulation parameters

$l_{m,j,1}$	267 time slots
u_G	3 time slots
u_B/u_G	25
$\lambda_B/(\lambda_B+\lambda_G)$	1/8
R_m	1 time slot
C_q	10
$Q^l_{m,j}/Q^r_{m,j}$	3
$P_{m,j}$	2

Table 4.1: Simulation parameter settings for online feasibility

For these results, deadline constraints were randomly generated such that $D_{m,j}$ is always between 1 and 1.5 times $l_{m,j,1}$, so that the local processing case is always feasible, but a variety of deadline constraints exist. This enables EDF to be productive, and allows us to compare its performance to the AQE algorithm.

One parameter value to note is R_m, which is set to 1 time slot; to say it another way, the server rate is set equal to the job size in units, resulting in a job only ever requiring 1 time slot at the server. This server rate is 3 times higher than the channel rate for advantaged users and more than 50 times faster than the channel rate for disadvantaged users. This allows us to compute a simplified version of the optimum offline schedule described in Chapter 2 by assuming the server is negligible (an assumption validated by the average server queue length of zero shown in Figure 4.2) and therefore ignoring server time slots and their constraints entirely. This simplification significantly reduces the overall number of constraints required to solve the optimization problem, as explained in section 2.3.1. By making this simplification, we have allowed the computation of that optimum offline schedule to become feasible in a reasonable amount of time.

The table of parameters includes u_B/u_G, the ratio of upload rates between disadvantaged and advantaged users (the very metric that causes disadvantaged users to be disadvantaged). As well, it provides $\lambda_B/(\lambda_B + \lambda_G)$ as 1/8, which means approximately 1/8 of all jobs generated are from disadvantaged users. In these results, where AQE is used, the channel queue cutoff is $C_q = 10$, for reasons described in section 4.2.2. Additionally, for these results $\max P_{m,j} = 2$, i.e. only the binary offloading case is considered; this is another simplification made to improve the feasibility of solving the ILP, and cases with more partition options are evaluated in section 4.2.2.

Maximum device class energy

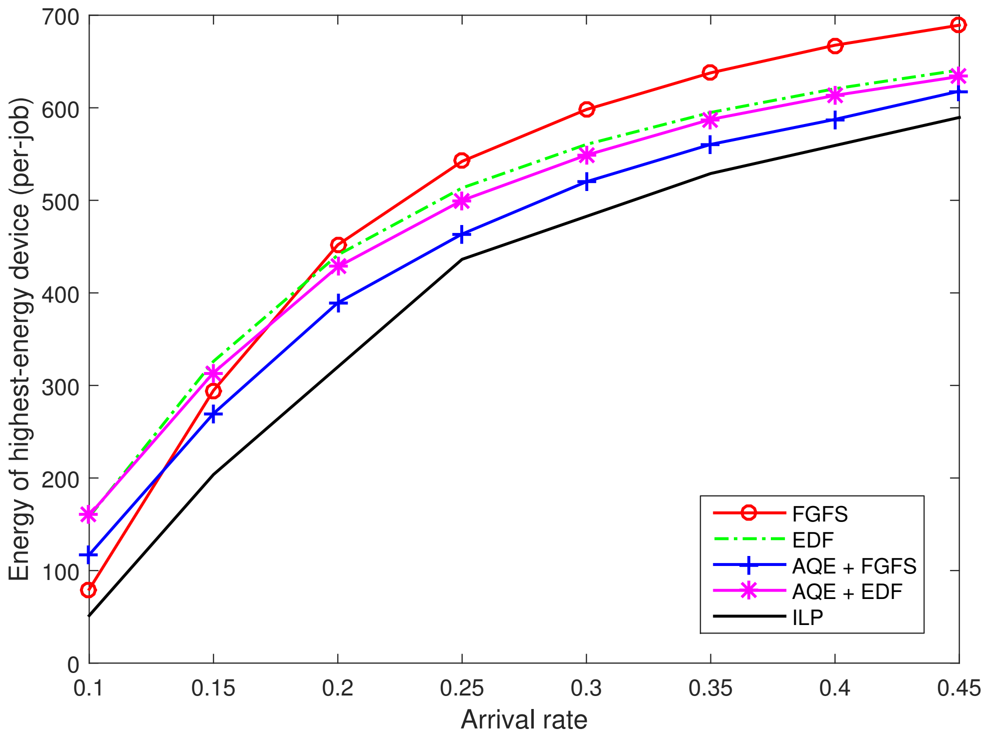

Figure 4.1: Maximum user class energy vs. job generation rate

In Figure 4.1 it can be seen that the ideal online scheduler is AQE used with FGFS

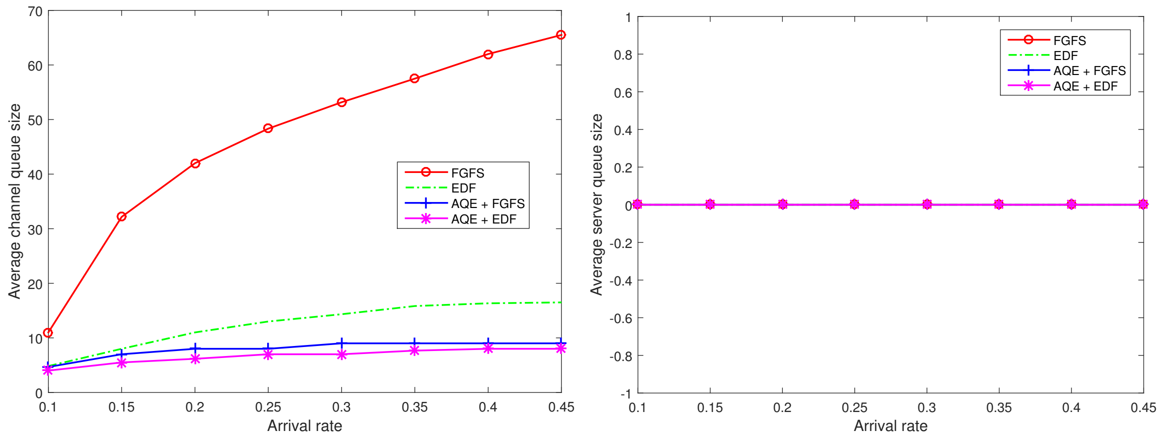

Figure 4.2: Average channel (left) and server (right) queue sizes

as the backing queue sorting method at every arrival rate above 0.1; note that only the energy of the user class with the highest per-job energy is compared here. Unlike other prioritization schemes that work towards energy fairness, AQE achieves this performance using only the information available from jobs currently in the queue, in real-time.

At the lowest arrival rate shown, the optimal bound demonstrates that it is still possible for disadvantaged users to have all jobs offloaded - while it may not be immediately obvious, the energy value indicated here by the optimal bound is the energy of offloading all units on the channel with no local processing at the parameters given, although no online scheduler achieves this.

At lower arrival rates, the AQE algorithm over-corrects slightly. Figure 4.2 demonstrates why, as the channel queue length at these arrival rates even without the AQE algorithm enabled is rarely long enough for the algorithm to have sufficient information to make an appropriate decision, and the parameters chosen have deliberately pushed the server queue to 0. However, the cutoff threshold specified prevents the AQE algorithm from performing significantly worse than doing nothing, and queue sizes rapidly grow large enough to be meaningful. As demonstrated in section 4.2.2, to increase the queue size threshold for improved performance at lower arrival rates would negate the advantages of using the algorithm at higher arrival rates. A similar effect explains the modest performance of AQE at all arrival rates when using EDF as the backing queue sorting method.

Energy fairness

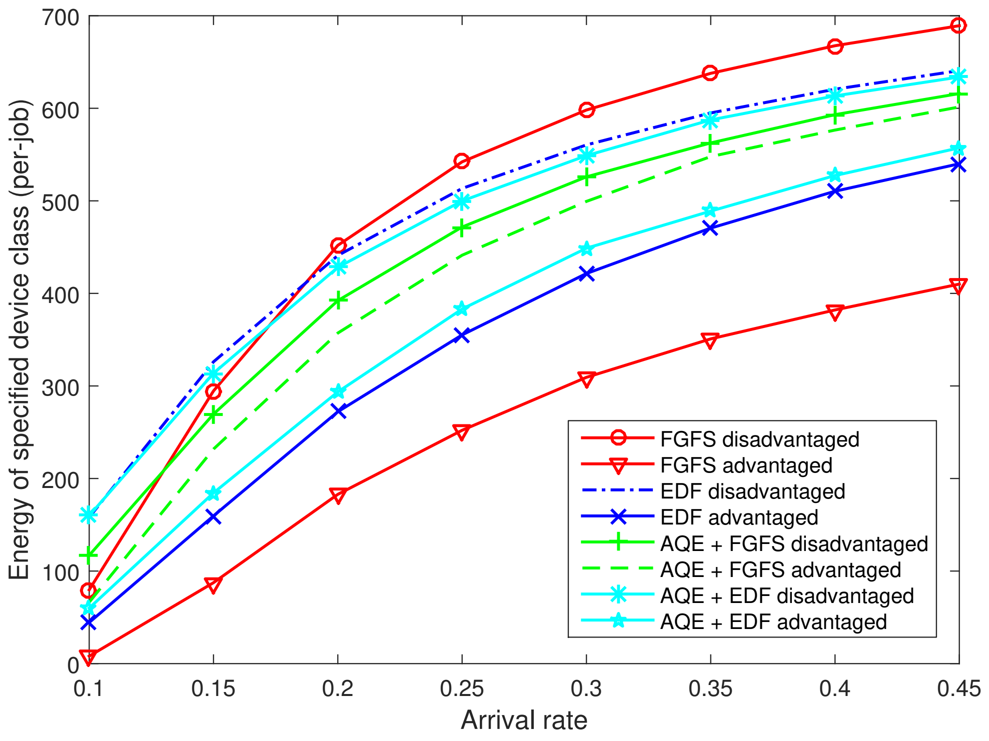

Figure 4.3: Comparison of online schedulers - Advantaged & disadvantaged user energy

Figure 4.3 compares the disadvantaged users' energy against the advantaged users' energy. (The optimal offline scheduling value is omitted here for clarity.) AQE with FGFS manages to reduce the difference in energy usage between advantaged and disadvantaged users to a value less than the absolute offloading energy disadvantage experienced by disadvantaged users, at all arrival rates. (This value is below 10% of the lower value for most arrival rates.)

Energy fairness - AQE vs. bound

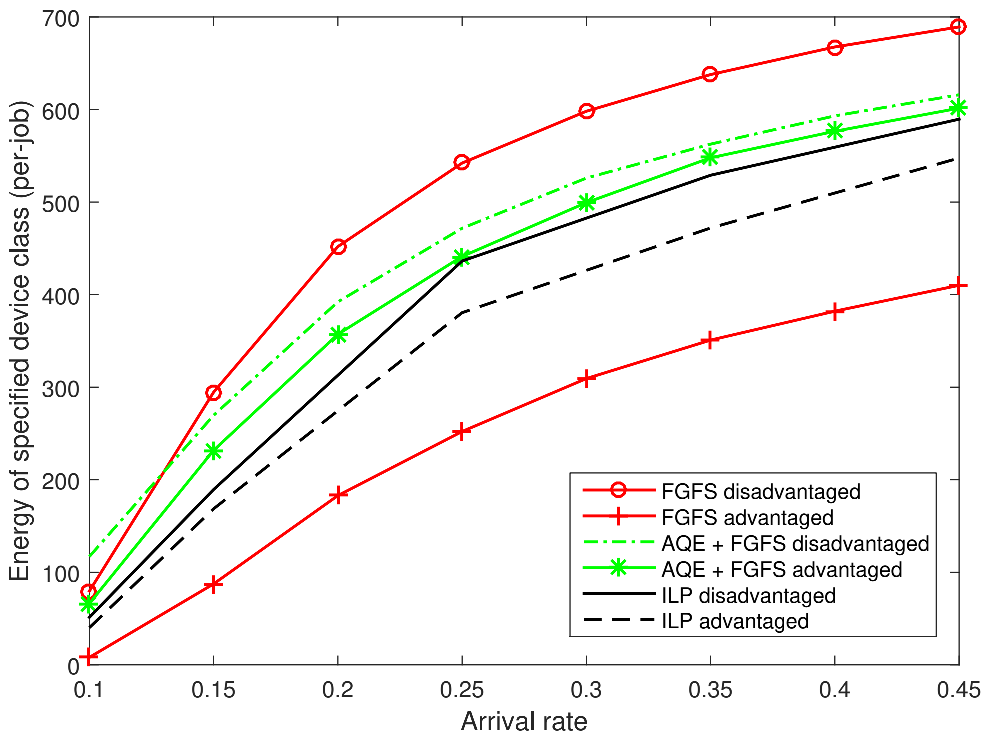

Figure 4.4: Comparison of AQE against bound

With the best online scheduler identified as AQE, the same plot as in Figure 4.3 is reproduced in Figure 4.4, this time with the redundant online schedulers removed, and the optimal offline bound added.

While the ILP does not explicitly work towards perfect energy fairness, it achieves something approaching fairness, and still serves as a lower bound on the individual disadvantaged and advantaged curves for AQE. The advantage of having all information about the system ahead of time is clear, although AQE still achieves a reasonably close result. FGFS is included on this plot as a comparison point in contrast to the algorithms that directly attempt to minimize the disparity between those two classes. It is clear that AQE achieves significantly better fairness than FGFS alone.

Energy fairness vs. overall average energy

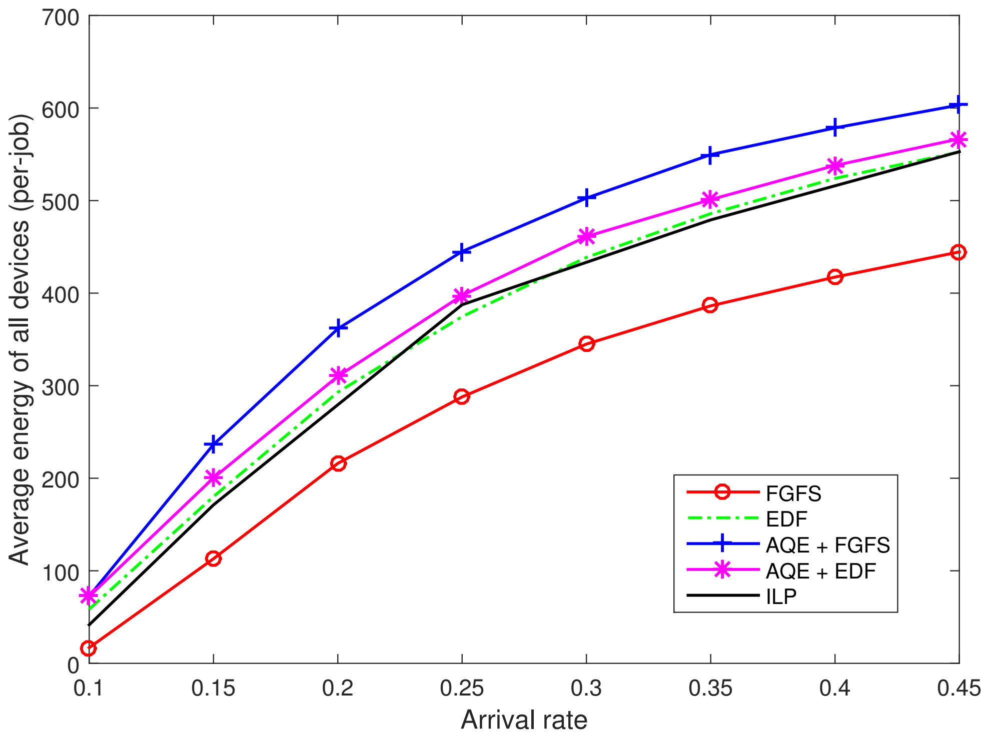

Figure 4.5: Comparison of online schedulers - Average user energy

Although it is the primary focus of this thesis, energy fairness is not the only criterion relevant to scheduling; there is a trade-off to be made between energy fairness and total average energy. Figure 4.5 demonstrates that where AQE (and even EDF) manage to outperform FGFS in terms of energy fairness (the difference in per-job energy between disadvantaged and advantaged user classes), this comes at the cost of average energy for all users, where FGFS significantly outperforms the other schedulers. The merits of choosing average energy versus energy fairness as scheduling criteria are potentially application-specific and are not investigated in detail in this thesis, but this plot shows that for online schedulers, one generally must be chosen ahead of the other, as demonstrated by the large distance between FGFS and even EDF.

Once again the bound produced by the ILP is only really a bound for the algorithms that attempt to maximize fairness, but serves as a bound here for those algorithms nonetheless.

4.2.2 Channel- and server-constrained results

With feasibility and efficacy established, it is worthwhile to further explore the AQE algorithm, demonstrating how its parameters are best selected and further exploring its strengths and weaknesses. The following results evaluate the performance of AQE with FGFS against merely using FGFS. These results were generated with the parameters listed in Table 4.2.

Simulation parameters

$l_{m,j,1}$	267 time slots
u_G	3 time slots
u_B/u_G	25
$\lambda_B/(\lambda_B+\lambda_G)$	1/8
R_m	5 time slots
$Q_{m,j}/Q_{m,j}$	3
$P_{m,j}$	5

Table 4.2: Simulation parameter settings for AQE performance evaluation

In this section, three significant changes to the parameters used in Section 4.2.1 are made. Firstly, to demonstrate that AQE performance remains high when both channel and server resources are constrained, the server rate is reduced to 50 from 267 in the previous section. This means the server is no longer negligible; unfortunately this change renders the optimal offline scheduling extremely difficult to solve as described in Chapter 2, so it has been omitted from these results. Secondly, partial offloading of jobs is now allowed by setting $\max p = 5$, which (as is also described in Chapter 2) allows jobs to be split between local and remote processing in multiples of $l_{m,j}/\max p$. This demonstrates that AQE is effective in more general offloading decision-making situations, in addition to the binary decision case. Thirdly, deadlines constraints are now fixed at $D_{m,j} = 1.25$. This is done to confirm that the advantage AQE obtains is not attained through any effect related to variable deadline constraints. This change renders EDF ineffective, as deadlines now differ solely based on arrival rate; however the removal of EDF from these results is inconsequential as the AQE algorithm is generally ineffective when used with it, as shown in previous results.

Energy fairness

Figure 4.6 demonstrates that AQE with FGFS remains highly effective, again keeping the difference between disadvantaged and advantaged users below the offloading energy disadvantage experienced by users with poor channel conditions, at all arrival rates.

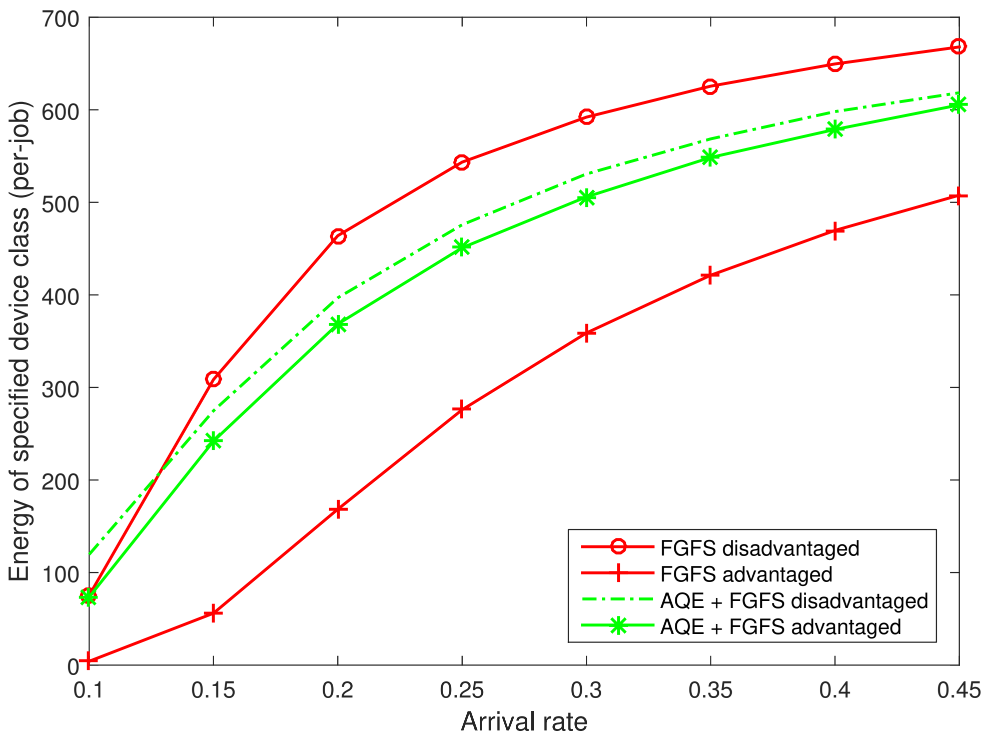

Figure 4.6: Advantaged vs. disadvantaged user energy - FGFS vs. AQE + FGFS

AQE queue size threshold

It is important to investigate the single configurable parameter related to the AQE algorithm, C_q, which is the queue size cutoff. Using the parameters set out for this section, Figure 4.7 demonstrates that the optimal queue size cutoff is somewhere around 2 times the inverse of the fraction of users that are disadvantaged. Below this point (represented by a cutoff of $C_q = 5$ on this graph) the algorithm will continue to maintain approximate energy parity, but overall energy will begin to grow significantly larger than even FGFS as altogether too many jobs are rejected. Alternately, above the ideal cutoff point, minimal gains - mostly at lower arrival rates - can be achieved by increasing the cutoff.

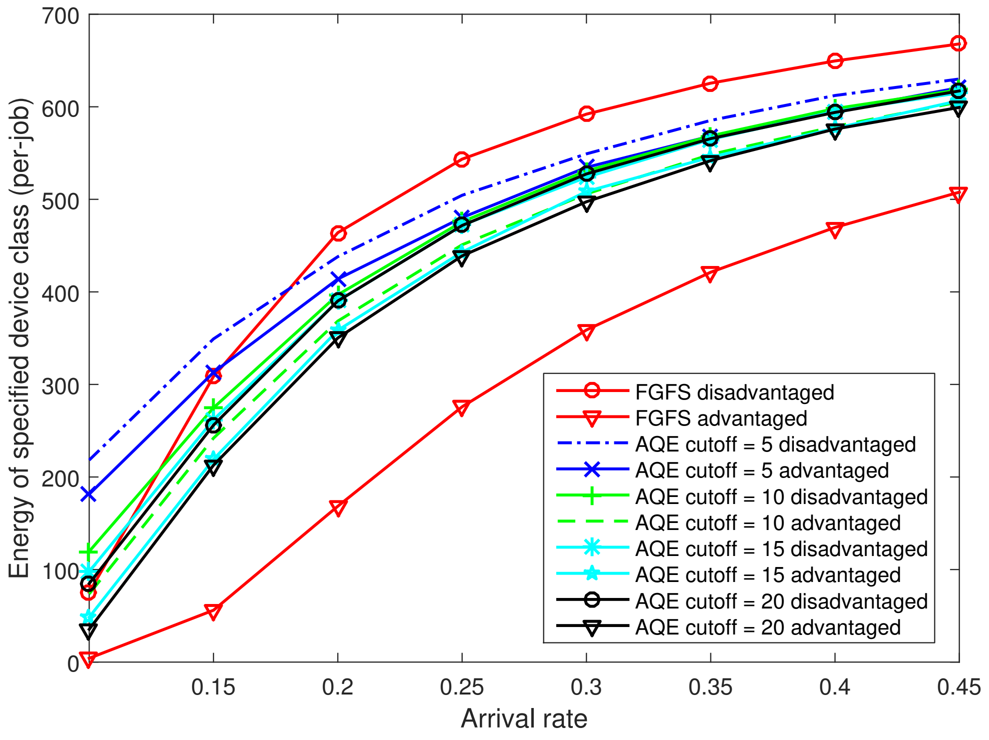

Figure 4.7: Energy with various values of C_q - 1/8 disadvantaged users

Choosing a low multiple of the inverse of the fraction of users that are disadvantaged is a reasonable benchmark for choosing a queue size cutoff; a queue with fewer than this number of jobs is much more likely to not contain any disadvantaged jobs to start with, which strongly influences the algorithm to reject any advantaged job arrivals until a disadvantaged job has been placed in the queue. If the threshold is very low, the queue is likely at any given time to contain no jobs whatsoever, at which point the algorithm erroneously determines that maintaining an empty queue is the most energy-neutral option, as discussed in Chapter 3.

In order to demonstrate this connection, the parameters from section 4.2.2 were used again, this time halving the frequency at which disadvantaged jobs are generated relative to advantaged jobs. Figure 4.8 shows the results of varying the queue size cutoff parameter C_q after making this modification.

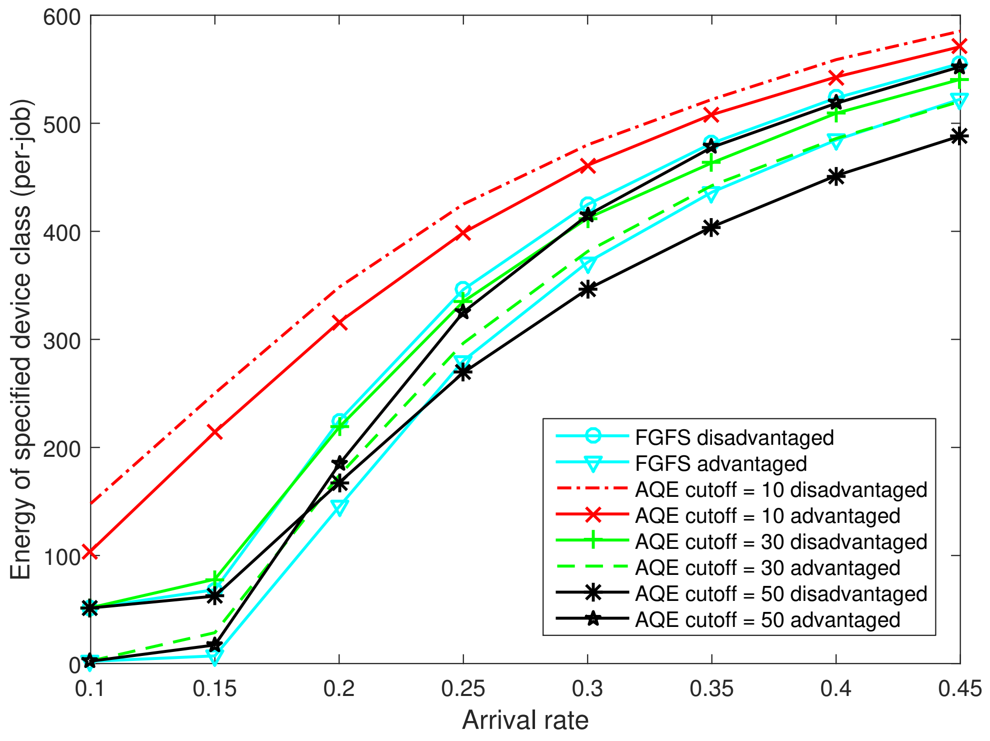

Figure 4.8: Energy with various values of C_q - 1/16 disadvantaged users

With these parameters the trend is further demonstrated; where AQE works well, its optimal queue size cutoff is somewhere around 2 times the inverse of the fraction of users that are disadvantaged. With that inverse fraction equal to 16 as it is in that figure, a queue size cutoff of $C_q = 10$ performs very poorly while a cutoff of $C_q = 30$ (roughly 2 times the inverse of the fraction) performs reasonably well. This figure also demonstrates an upper limit on the queue size cutoff; with these parameters, by a cutoff of $C_q = 50$, the algorithm is already significantly overcompensating at most arrival rates.

4.3 Summary

In this section the efficacy of the AQE algorithm was demonstrated across several combinations of parameters representing various scenarios, all of which demonstrated that AQE achieves the best energy fairness out of the online scheduling algorithms examined. It achieves this by making decisions about job acceptance and prioritization based only on the information available from the current job queue. This was demonstrated against an optimal offline schedule as a lower bound on the online schedulers' performance. The trade-off between energy fairness and average energy for all users was also demonstrated. Finally, further work was done to explore the ideal queue size cutoff for applying AQE, where it was shown the ideal value is connected to the frequency of disadvantaged jobs relative to advantaged jobs.

Chapter 5

Conclusions and Future Work

This thesis began by introducing the area of mobile cloud computing, including its advantages, disadvantages, challenges and potential. In particular, a volume of existing work in the field was described and evaluated to provide context to the rest of the thesis.

Following this introduction, the system model used throughout the thesis was introduced and detailed. This model provided a time slot-based implementation of a constrained shared channel used to upload to a similarly-constrained shared server. It defined the energy consumption required for processing jobs subject to deadline constraints, within which a fixed amount of processing must be completed using some combination of local and remote processing resources. The mechanisms for selecting the amount of offloading to perform and the time slot schedules used to perform it were also defined.

With the system model defined, an offline scheduler was implemented as an integer linear optimization problem to provide an optimal offline min-max energy schedule for a given set of jobs defined within that model. The constraints on this objective were described in detail, and the complexity of the problem as NP-complete proven.

In addition to the optimal offline scheduler, several online scheduling algorithms were also defined and implemented. Once a method was described for determining the feasibility of offloading a given job, the schedulers were defined in terms of that feasibility. In addition to First Generated First Served and Earliest Deadline First, two algorithms based off of basic and traditional scheduling methods used extensively, the Average Queue Energy Prioritization scheduling method was introduced and described. The other online scheduling methods were used as comparison points to demonstrate its features.

These scheduling methods were then compared using a variety of parameter settings and in numerous ways, through solving of the offline optimization problem, and through simulation of the online schedulers. These results demonstrated that the AQE algorithm outperforms the other online scheduling algorithms evaluated, by coming closest to the energy fairness of the offline schedules generated by the solution of the offline optimization problem. It achieves this using nothing more than the information available from the jobs currently accepted and waiting for processing in the queue.

There exist several opportunities to expand on this work in the future. Extending this model further, to include multiple shared channels and/or servers, may provide a number of potential avenues for interesting work; such work could include multiple shared channels which are available to all devices but have different characteristics, or a similar scenario involving the shared servers, for which an energy-fair allocation of the users accessing those resources is still a primary concern.

Another potential avenue for such future exploration is the addition of non-negligible download requirements following server processing. It is assumed throughout this thesis that upload requirements are far greater than download requirements, making download requirements negligible; in an additional extension of this model, it could be the case that download requirements are large enough to impact overall energy and offloading feasibility. These additional download requirements might be processed by a separate download channel, either shared or unshared, or potentially would need to be fulfilled by using time slots from the shared upload channel, further complicating the system model and providing additional constraints on the schedulers.

An additional opportunity for further investigation is in scenarios where some users experience unreliable connections which prevent successful offloading even when offloading would be feasible and desirable. Mitigation strategies, including anticipation of poor channel conditions, acceptance of the partial offloading achieved before connection failure, and preemptive simultaneous local and remote processing, are potential candidates for extensions of this work.

Bibliography

Achary, R., Vityanathan, V., Raj, P., and Nagarajan, S. (2015). Dynamic Job Scheduling Using Ant Colony Optimization for Mobile Cloud Computing. In R. Buyya and S. M. Thampi, editors, *Intelligent Distributed Computing*, number 321 in Advances in Intelligent Systems and Computing, pages 71–82. Springer International Publishing. DOI: 10.1007/978-3-319-11227-5_7.

Adams, J., Balas, E., and Zawack, D. (1988). The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, **34**(3), 391–401.

Aguayo, D., Bicket, J., Biswas, S., Judd, G., and Morris, R. (2004). Link-level Measurements from an 802.11b Mesh Network. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 121–132, New York, NY, USA. ACM.

Arthur, C. (2014). Your smartphone's best app? Battery life, say 89% of Britons. *The Guardian*.

Barbarossa, S., Sardellitti, S., and Lorenzo, P. D. (2013). Joint allocation of computation and communication resources in multiuser mobile cloud computing. In *2013*

IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC), pages 26–30.

Barbera, M. V., Kosta, S., Mei, A., and Stefa, J. (2013). To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In *2013 Proceedings IEEE INFOCOM*, pages 1285–1293.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. (2011). CloneCloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM.

Cisco (2015). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014–2019 - Cisco-Visual-Networking-Index-Global-Mobile-Data-Traffic-Forecast-Update-2014–2019.pdf.

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

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

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

George, D., Stryjak, J., Meloán, M., and Castells, P. (2016). The Mobile Economy 2016. Technical report, GSMA Intelligence.

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

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*, MCS '10, pages 6:1–6:5, New York, NY, USA. ACM.

Johnson, B. (2013). How Siri Works. *http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/siri.htm*, page 3.

Kemp, R., Palmer, N., Kielmann, T., and Bal, H. (2010). Cuckoo: a computation offloading framework for smartphones. In *International Conference on Mobile Computing, Applications, and Services*, pages 59–79. Springer.

Khan, A. N., Mat Kiah, M. L., Khan, S. U., and Madani, S. A. (2013). Towards secure mobile cloud computing: A survey. *Future Generation Computer Systems*, **29**(5), 1278–1299.

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

Klein, A., Mannweiler, C., Schneider, J., and Schotten, H. D. (2010). Access Schemes for Mobile Cloud Computing. In *2010 Eleventh International Conference on Mobile Data Management*, pages 387–392.

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. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE.

Kruk, Ł., Lehoczky, J., Shreve, S., and Yeung, S.-N. (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, B., Li, J., Huai, J., Wo, T., Li, Q., and Zhong, L. (2009). EnaCloud: An Energy-Saving Application Live Placement Approach for Cloud Computing Environments. In *IEEE International Conference on Cloud Computing, 2009. CLOUD '09*, pages 17–24.

Li, B., Liu, Z., Pei, Y., and Wu, H. (2014). Mobility Prediction Based Opportunistic Computational Offloading for Mobile Device Cloud. In *2014 IEEE 17th International Conference on Computational Science and Engineering (CSE)*, pages 786–792.

Lin, X., Wang, Y., Xie, Q., and Pedram, M. (2015). Task Scheduling with Dynamic Voltage and Frequency Scaling for Energy Minimization in the Mobile Cloud Computing Environment. *IEEE Transactions on Services Computing*, **8**(2), 175–186.

Liu, Q., Jian, X., Hu, J., Zhao, H., and Zhang, S. (2009). An Optimized Solution for Mobile Environment Using Mobile Cloud Computing. In *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–5.

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.

Miettinen, A. P. and Nurminen, J. K. (2010). Energy Efficiency of Mobile Clients in Cloud Computing. *HotCloud*, **10**, 4–4.

Namboodiri, V. and Ghose, T. (2012). To cloud or not to cloud: A mobile device perspective on energy consumption of applications. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pages 1–9.

Rahimi, M. R., Ren, J., Liu, C. H., Vasilakos, A. V., and Venkatasubramanian, N. (2013). Mobile Cloud Computing: A Survey, State of Art and Future Directions. *Mobile Networks and Applications*, **19**(2), 133–143.

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*, DAC '03, pages 906–911, New York, NY, USA. ACM.

Rudenko, A., Reiher, P., Popek, G. J., and Kuenning, G. H. (1998). Saving Portable Computer Battery Power Through Remote Process Execution. *SIGMOBILE Mob. Comput. Commun. Rev.*, **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*, SAC '99, pages 365–372, New York, NY, USA. ACM.

Sanaei, Z., Abolfazli, S., Gani, A., and Buyya, R. (2014). Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges. *IEEE Communications Surveys & Tutorials*, **16**(1), 369–392.

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 D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, number 1459 in Lecture Notes in Computer Science, pages 180–198. Springer Berlin Heidelberg. DOI: 10.1007/BFb0053987.

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 (IWQoS)*, pages 284–289.

Valery, O., Chou, J. C., Tsao, Y., Liu, P., and Wu, J. J. (2015). A Partial Workload Offloading Framework in a Mobile Cloud Computing Context. In *2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 43–50.

Wan, Z., Wu, J., and Zheng, H. (2015). Utility-Based Uploading Strategy in Cloud

Scenarios. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8.

Wang, S. and Dey, S. (2013). Adaptive Mobile Cloud Computing to Enable Rich Mobile Multimedia Applications. *IEEE Transactions on Multimedia*, **15**(4), 870–883.

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*, volume 2, pages 1–8.

Xu, Y. and Mao, S. (2013). A survey of mobile cloud computing for rich media applications. *IEEE Wireless Commun.*, **20**(3), 1–0.

Yang, L., Cao, J., Yuan, Y., Li, T., Han, A., and Chan, A. (2013). A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing. *SIGMETRICS Perform. Eval. Rev.*, **40**(4), 23–32.

Yue, J. (2015). *Energy Fair Cloud Server Scheduling in Mobile Computation Offloading*. Master's thesis, McMaster University.

Yue, J., Zhao, D., and Todd, T. (2014). Cloud server job selection and scheduling in mobile computation offloading. In *2014 IEEE Global Communications Conference (GLOBECOM)*, pages 4990–4995.

Zhang, W., Wen, Y., and Wu, D. O. (2013a). Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In *2013 Proceedings IEEE INFOCOM*, pages 190–194.

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

Zhang, W., Wen, Y., and Wu, D. (2015). Collaborative Task Execution in Mobile Cloud Computing Under a Stochastic Wireless Channel. *IEEE Transactions on Wireless Communications*, **14**(1), 81–93.