Resource Allocation in Mobile Edge Cloud Computing for Data-Intensive Applications

by

Mohammad Nour ALKHALAILEH

Supervisor: Assoc. Prof. Bahman Javadi

Associate Supervisors: Dr. Rodrigo N. Calheiros and Dr. Quang Vinh Nguyen

A thesis submitted in total fulfillment for the degree of Doctor of Philosophy

in the

School of Computer, Data and Mathematical Sciences

Western Sydney University

February 2021

Dedication

This Thesis is dedicated to my parents

In memory of my father

In memory of my mother

With love and eternal appreciation for their endless love, support and encouragement

Acknowledgements

The completion of this thesis would not have been possible without the support and encouragement of several special people. I would like to take this opportunity to show my gratitude to those who have assisted me in a myriad of ways.

First and foremost, I would like to express my deep and sincere gratitude and heartfelt thanks to my research supervisor A/Prof. Bahman Javadi for his continuous support of my PhD study and research, for his patience, motivation, enthusiasm, and generosity with his immense knowledge. His guidance was invaluable to me during all stages of my research and when writing this thesis.

I could not imagine a better advisor and mentor for my PhD study. Dr Bahman is someone you will instantly love and never forget once you meet him. He's the funniest advisor and one of the smartest people I know. I hope that I can learn to be as lively, enthusiastic and energetic as Dr Bahman and to someday command an audience as well as he can.

I would also like to express my appreciation to my co-supervisors Dr Rodrigo Calheiros and Dr Quang Vinh Nguyen for their encouragement, insightful comments and great help with many aspects of my research.

I thank to my close friend, Raed Alsurdeh, for helping me to solve technical issues and supporting me during my research journey. Great thanks to my research team, Bilal, Yogesh, Rekha and friends. Thank you, Nabil and Guang, for five years of technical computing support. Thank you to all staff in the library and GRS for all help and support I received. Thank you to Dr Campbell Aitken, who provided professional editing services in accordance with the Institute of Professional Editors' *Guidelines for editing research theses*.

I would like to express my sincere gratitude to Western Sydney University. WSU has assisted me in gaining new research skills and knowledge of IT over the last five years.

I am also thankful for Australia for hosting me for the last 12 years. I have come to realise a lot about education, life, reality and most of all my destiny and journey in this life.

I am indebted to my beloved parents for their continuous support. I would not be the person I am today without their tremendous help and sacrifice. I owe all my achievements and success to them.

Last but not least, I would like to thank my family: my lovely wife Noor, my kids (Yousef, Zain and Batool), my brothers, my sister, my Parent-in-law and my brothers-in-law for supporting me spiritually throughout writing this thesis and my life in general.

Mohammad ALKHALAILEH

July 2020

Declaration of Authorship

I, Mohammad Nour ALKHALAILEH, declare that this thesis titled, Resource Allocation in Mobile Edge Cloud Computing for Data-Intensive Applications and the work presented in it are my own.

I confirm that the work presented in this thesis is, to the best of my knowledge and belief, original except as acknowledged in the text. I hereby declare that I have not submitted this material, either in full or in part, for a degree at this or any other institution.

Signed 

Date: 08 / 02 / 2021

Contents

List of Figures

List of Tables

Abbreviations

API	**A**pplication **P**rogram **I**nterface
BB	**B**ranch and **B**ound
BoT	**B**ag **o**f **T**asks
CI	**C**onfidence **I**nterval
CPU	**C**entral **P**rocessing **U**nit
DAG	**D**irected **A**cyclic **G**raph
ETSI	**E**uropean **T**elecommunications **S**tandards **I**nstitute
GPS	**G**lobal **P**ositioning **S**ystem
IaaS	**I**nfrastructure **a**s **a** **S**ervice
ILP	**I**nteger **L**inear **P**rogramming
I/O	**I**nput/**O**utput
IoT	**I**nternet **o**f **T**hings
IoV	**I**nternet **o**f **V**ehicles
JSON	**J**ava**S**cript **O**bject **N**otation
LP	**L**inear **P**rogramming
MC	**M**obile **C**omputing
MCC	**M**obile **C**loud **C**omputing
MEC	**M**obile **E**dge **C**omputing
MECC	**M**obile-**E**dge **C**loud **C**omputing
MIPS	**M**illion **I**nstructions **P**er **S**econd
MILP	**M**ixed **I**nteger **L**inear **P**rogramming
MINLP	**M**ixed **I**nteger **N**on**L**inear **P**rogramming
PaaS	**P**latform **a**s **a** **S**ervice
PSO	**P**article **S**warm **O**ptimization

QoE	**Q**uality **o**f **E**xperience
QoS	**Q**uality **o**f **S**ervices
REST	**R**epresentational **S**tate **T**ransfer
SaaS	**S**oftware **a**s **a** **S**ervice
SDN	**S**oftware **D**efined **N**etworking
VD-ABC	**V**ariety-**D**riven **A**rtificial **B**ee **C**olony
VM	**V**irtual **M**achine
WAP	**W**ireless **A**pplication **P**rotocol
WiFi	**W**ireless **F**idelity
3G	**T**hird **G**eneration
4G	**F**ourth **G**eneration

Western Sydney University

Abstract

School of Computer, Data and Mathematical Sciences

Doctor of Philosophy

by Mohammad Nour ALKHALAILEH

Rapid advancement in the mobile telecommunications industry has motivated the development of mobile applications in a wide range of social and scientific domains. However, mobile computing (MC) platforms still have several constraints, such as limited computation resources, short battery life and high sensitivity to network capabilities. In order to overcome the limitations of mobile computing and benefit from the huge advancement in mobile telecommunications and the rapid revolution of distributed resources, mobile-aware computing models, such as mobile cloud computing (MCC) and mobile edge computing (MEC) have been proposed. The main problem is to decide on an application execution plan while satisfying quality of service (QoS) requirements and the current status of system networking and device energy. However, the role of application data in offloading optimisation has not been studied thoroughly, particularly with respect to how data size and distribution impact application offloading. This problem can be referred to as data-intensive mobile application offloading optimisation. To address this problem, this thesis presents novel optimisation frameworks, techniques and algorithms for mobile application resource allocation in mobile-aware computing environments. These frameworks and techniques are proposed to provide optimised solutions to schedule data intensive mobile applications. Experimental results show the ability of the proposed tools in optimising the scheduling and the execution of data intensive applications on various computing environments to meet application QoS requirements. Furthermore, the results clearly stated the significant contribution of the data size parameter on scheduling the execution of

mobile applications. In addition, the thesis provides an analytical investigation of mobile-aware computing environments for a certain mobile application type. The investigation provides performance analysis to help users decide on target computation resources based on application structure, input data, and mobile network status.

Chapter 1

Introduction

In recent years mobile telecommunications has become an important part of life and business via improving methods of social communication, e-commerce, smart applications and so on. The number of users continues to increase and the volume of data generated from users' devices has grown exponentially.

According to a recent report [1], by 2022, mobile data traffic is expected to reach 77.5 exabytes per month worldwide at a compound annual growth rate of 46%. Nearly two-thirds of the global population will have Internet access by 2022, with an estimated 5.3 billion total users' up from 3.9 billion (51% of global population) in 2018. Moreover, it is predicted that by 2023, the fourth generation of broadband cellular network technology (4G) will be responsible for 54.3% of total mobile connections, compared to 34.7% in 2017, and connections will grow from 3 billion in 2017 to 6.7 billion by 2022 [1]. The advances in technology and the broad uptake of mobile applications in many community and business sectors, such as social media, healthcare and education, indicate potential to introduce more applications into other environments. Mobile e-business applications, for example, require highly scalable resources with high security measures to support communication-intensive customer requests. Mobile gaming can benefit from cloud computing scalability and high computation power to process complex gaming scenarios. Mobile devices have inherent limitations, however, pertaining to processing data-intensive tasks

such as information discovery, image processing and a reliance on battery power as a source of energy.

The use of mobile technologies (mobile hardware, software, and communication) to create, process and store information is generically described as mobile computing (MC) [2]. Mobile applications improve information accessibility, integrate technologies with information systems, and increase management effectiveness. Nowadays, a wide range of mobile applications are data-driven, tending to use huge amounts of mobile data due to their demand for high computation power and memory space [3]. Data-intensive applications, such as customised data analytic services, natural language processing and face recognition, are resource-intensive, requiring machines with powerful central processing units (CPU) and huge memory space to load dynamic application code and data. For example, the computation for intelligent assistant applications such as Siri (Apple), Now (Google), and Cortana (Microsoft) is performed in the cloud; this requires significant amounts of data to be migrated via wireless networks and creates substantial computational load on data centres [4]. A general term for overcoming limitations of mobile devices is known as "computation offloading", which implies migrating intensive computations from resource-limited devices (mobile devices) to resource-rich machines such as clouds, cloudlets, and edges to minimise the execution time and optimise the device energy [5–7]. Computing offloading involves distributing computation over a computing system to benefit from local and external capabilities. The next section describes the foundation of mobile-aware computing models employed to support the mobile offloading process.

1.1 Mobile-aware Computing Models

Computation and data offloading is an essential tool to enable smart devices to deliver rich applications. Understanding the features and requirements of mobile-aware computing systems is essential to develop efficient offloading strategies and

techniques to meet the requirements of diverse application models and QoS. This section provides a brief illustration of some mobile-aware computing systems.

1.1.1 Mobile Computing

The term "mobile computing" implies that the resources of a mobile device can be used without consideration of physical location or network connectivity. MC was first introduced in the early 1990s in association with ubiquitous computing [8]. Weiser [8] described the scenarios and applications of mobile-based hardware architectures and wireless communications. The MC vision is built on the combination of three concepts: wireless communication, mobility, and portability [9, 10].

- *Wireless communication.* Wireless communication technology is the essential enabler of MC; it allows communication between mobile system components for data transmission and Internet access. Wireless communication connectivity and quality are fundamental issues for MC system capability.

- *Mobility.* Mobility features enable smartphones to have transparent access to computing infrastructure and shared resources. Device mobility supports mobile applications in different ways. It improves information accessibility, with the level of improvement depending on mobile hardware and communication efficiency. MC allows organisations to integrate technologies with existing information systems, but has limitations. For instance, mobile technologies, such as general packet radio service, enhanced data rates for global system for mobile communication evolution, and the third generation of wireless mobile telecommunications technology (3G) support insufficient network bandwidth, have inadequate resources for processing intensive tasks and is energy inefficient [2].

The mobile device scheme of communication is governed by wireless communication protocols. One of the leading wireless protocols is wireless application protocol (WAP) [11]. WAP offers a common environment to develop value-added services for mobile devices. The protocol is an open standard that abstracts network infrastructures, hardware profiles, and transport mechanisms in a scalable fashion [12]. Nevertheless, mobile network availability and stability, which are not guaranteed, are critical features for sustainable device participation and engagement. Moreover, the uncertainty of device location makes it challenging to predict critical offloading performance parameters, such as network latency and device responsiveness [13].

- *Portability.* Portability is the core enabler of device mobility. The ultimate challenge for portability is the unavailability of a continuous power source. Mobile devices rely on short-term, low-capacity power supplies (batteries). Networking services and mobile processes are energy-hungry, particularly in the case of smart devices. Energy management is a complicated task and has a significant influence on mobile device performance as a component of a computing model.

1.1.2 Hybrid Mobile Cloud Computing

Cloud computing provides computation, storage and telecommunication services in a scalable and virtualised manner usually as scalable services and under the "pay-as-you-go" business model [14–17]. Cloud computing provides computation and storage infrastructure and services for users to implement and host applications with minimal configuration and setup complexity. The most common cloud services are Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Cloud computing has been adopted to resolve the limitations of mobile computing, and in this case is known as mobile cloud computing (MCC) [3]. Figure 1.1 illustrates a high-level architecture for MCC. The core service of MCC is deriving benefits from cloud capabilities, such as vast storage, high processing power, provisioning and scalability in running

data-intensive applications. Many of MCC frameworks have been proposed in the literature [11, 12, 14–24]; most target computation-intensive applications and employ offloading techniques to enhance the performance of the MCC framework by distributing or migrating intensive tasks to powerful servers in the cloud [25]. Two primary offloading techniques described in the literature are code offloading and remote execution. Code offloading involves application-code partitioning, and remote execution adopts task-oriented mobile web services as an offloading strategy.

In addition to the features inherited from MC, discussed in the previous section, MCC comprises additional features of resource heterogeneity, sociability, accessibility and security. MCC architecture consists of three elements, namely, mobile devices, remote clouds, and networking. Understanding how these elements are integrated and collaborate is essential for the construction of efficient computation and data offloading systems.

- *Mobile Devices.* Mobile devices are entities capable of performing lightweight computation and storage services. MCC aims to augment mobile devices by offloading computation and data-intensive jobs to more powerful entities like

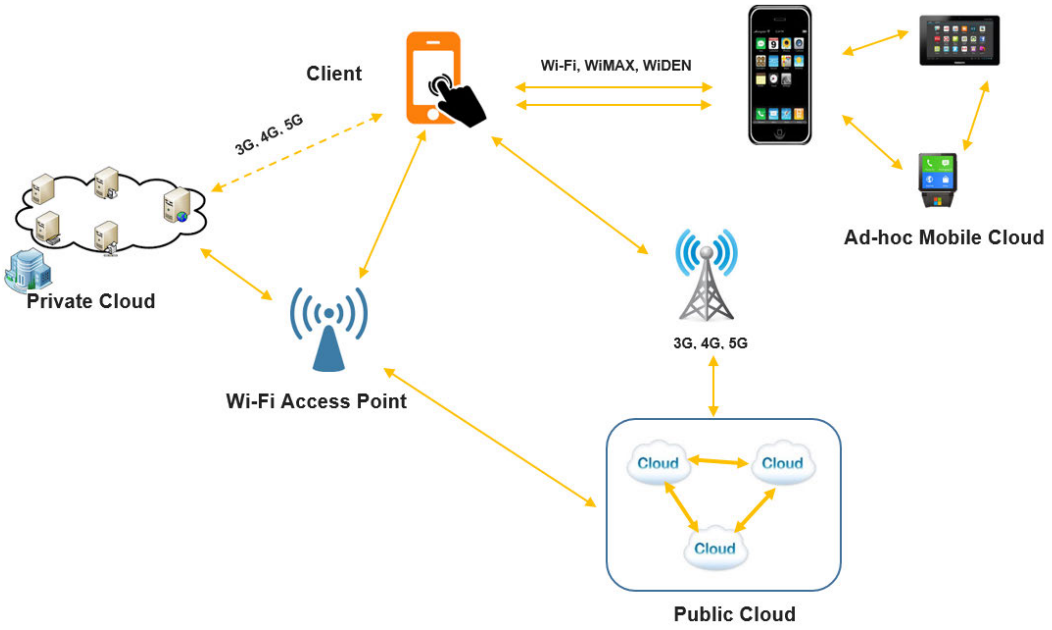


FIGURE 1.1: Mobile cloud computing general architecture

cloudlets and remote clouds. However, in some cases, such as low network quality, high data transmission cost and high dependency on device features, local execution is the preferred option.

- *Remote Cloud.* The remote cloud or Internet cloud basically represents the highest level of computation power on the MCC model. It receives offloading requests, which are either indirectly forwarded from cloudlets or directly forwarded by mobile devices. Satyanarayanan et al. [24] proposed that the cloud computing model is the best solution for solving the deficiencies of mobile device resources. Cloud computing is a promising integral component to assist with mitigating problems of the MC model, for many reasons. First, cloud computing can extend mobile device battery life by migrating complex processing to more powerful servers, thus minimising the total execution time on mobile devices. Second, cloud computing enables mobile users to store/access large data stores on the cloud via a wireless communication protocol. For example, cloud computing supports mobile users with image exchange services, which save considerable energy and storage space, as all images are sent to the cloud and processed on the cloud. Third, cloud resources can be scaled to meet unpredictable mobile application development demands. Lastly, the availability of different cloud service providers makes it feasible to integrate cloud services to satisfy users' demands.

- *Networking.* Networking is the most critical part of a mobile-aware computing model. The feasibility of undertaking the offloading depends on networking availability and quality. The networking interface or protocol also has significant influence on the offloading decision and process. For example, cellular and mobile networks are expensive and consume more energy than wireless fidelity (WiFi) networks [26].

Mobile cloud computing can support many application types, including commerce, multimedia sharing, gaming, banking, social sensing, traffic monitoring and healthcare assistance [27]. For example, in MCC-based learning applications, thin clients (mobile applications) can access cloud-based learning information at any time and

in any place. Mobile sensor features enable MCC applications to collect sensing information; the domains of this type of application might encompass online diagnoses, health monitoring and social networking. For instance, customers may store and access their profile data regardless of time and geography constraints. For social networking applications, MCC allows data and multimedia sharing in real time across services. Another example is a mobile-based healthcare system in a public cloud environment [28]. The system collects data from patients through attached sensors. Data is sent immediately to the cloud to be processed by sophisticated analytic tools. The major contributions of the cloud are minimising the communication overheads between all actors in the system and having all system data stored in a centralised, highly accessible yet secure repository.

The next generation of techniques of offloading optimisation was developed as a response to the increasing popularity of MC in many aspects of human life. Cloud computing effectively embraces benefits from scalability and sustainability solutions. MCC offers opportunities to overcome energy limitations to handle computation-intensive tasks and is able to offload heavy workloads to the cloud, benefiting from its high computation capabilities to reduce the overhead of running these tasks locally in user mobile devices.

The main shortcoming of MCC architecture is the necessity of transferring large data files over mobile networks, which brings significant challenges with respect to the monetary cost and energy involved in data transfer. To overcome these issues, the cloudlet-based computing concept was proposed. Cloudlets are "decentralised and widely dispersed Internet infrastructure components whose compute cycles and storage resources can be leveraged by nearby mobile computers" [24].

Cloudlet features of self-management, Internet connectivity, and moderate capability make it a suitable resource model for many application scenarios. The cloudlet-based resource model provides services like local resource management, connecting local devices, executing offloading requests, and forwarding offloading tasks to remote cloud resources in case of insufficient processing capabilities to process incoming workloads. According to the granularity and complexity

of offloading requests, the cloudlet system can determine the offloading schedule and decides which tasks are executed locally or migrated to the cloud systems. The integration between cloudlet-based and MCC models is referred to herein as hybrid MCC, a concept which integrates two or more MCC basic architectures (CloneCloud, cloudlet, and mobile cloud). The main objective of this integration is to enhance overall application performance by utilising the benefits of each integrated model. A clone hosted on a public cloud can be a high-performance server, while a cloudlet is deployed via different wireless communication protocols.

The main problem facing a hybrid MCC integration scheme is guaranteeing seamless interaction between them, allowing the elimination of data duplication, providing high service accessibility and reducing the complexity of application migration [29]. Edge resources allow efficient solutions and offer computation and storage resources closer to user devices [30].

1.1.3 Mobile-Edge Cloud Computing

With the spread of Internet of Things (IoT) enabled and smart applications, such as virtual reality [31], smart grids [32], and smart environments [33], the move towards delay-sensitive applications poses a challenge for MCC architecture to meet delay requirements. The MCC paradigm has limitations in its ability to meet the requirements of low latency, location awareness and mobility support [34, 35]. Thus, edge computing is becoming a popular way to replicate cloud functions [36]. Edge computing brings the computation, data, applications, and services the network edge and reduces dependency on cloud servers[38]. Mobile edge computing (MEC) combines the capabilities of mobile edge networks and the Internet to divert latency-sensitive services to the edge of mobile networks, thus enhancing the various types of service quality. MEC is broadly applied in many application contexts for highly responsive, secure, real-time and latency-sensitive services[39].

The major limitation of edge devices' computation capability is their physical limitations, which make it challenging to handle computation and data-intensive tasks [40]. Moreover, MEC is not an appropriate computing model for handling large-scale applications that involve migrating high computation and data workloads. To overcome the physical limitations of edge resources, MEC and MCC architectures can be combined to benefit from the high capability of cloud servers and low-latency processing at the edge layer. Integrating edge and cloud provides benefits from the high capability of cloud resources and the availability of resources at the edge layer. This joint computation architecture can be referred to as mobile-edge cloud computing (MECC) [41]. Figure 1.2 shows an example of MECC architecture in an integrated computation system of edge and cloud resources. Application/task offloading techniques can overcome QoS constraints by considering the opportunities of each computation system model.

For instance, simple computational tasks can be processed locally or at the edge layer to preserve device energy and achieve response with low latency. For computation-intensive tasks, edge cloud collaboration can be applied to benefit from the capabilities of cloud servers while meeting latency and energy consumption constraints [41].

1.2 Computation Offloading

Computation offloading refers to the concept of boosting mobile systems by migrating complex computation to highly capable computers [42]. The offloading process relies on the decision of running a task locally (on the user device) or remotely (external server). The decision is based on the task complexity, user device capability and energy status, and mobile cloud environment, such as scalability, availability, energy and cost-efficiency [45]. Computation offloading in the context of mobile-cloud application processing follows three main techniques: application partitioning, code offloading and remote execution [24] [28, 43, 46].

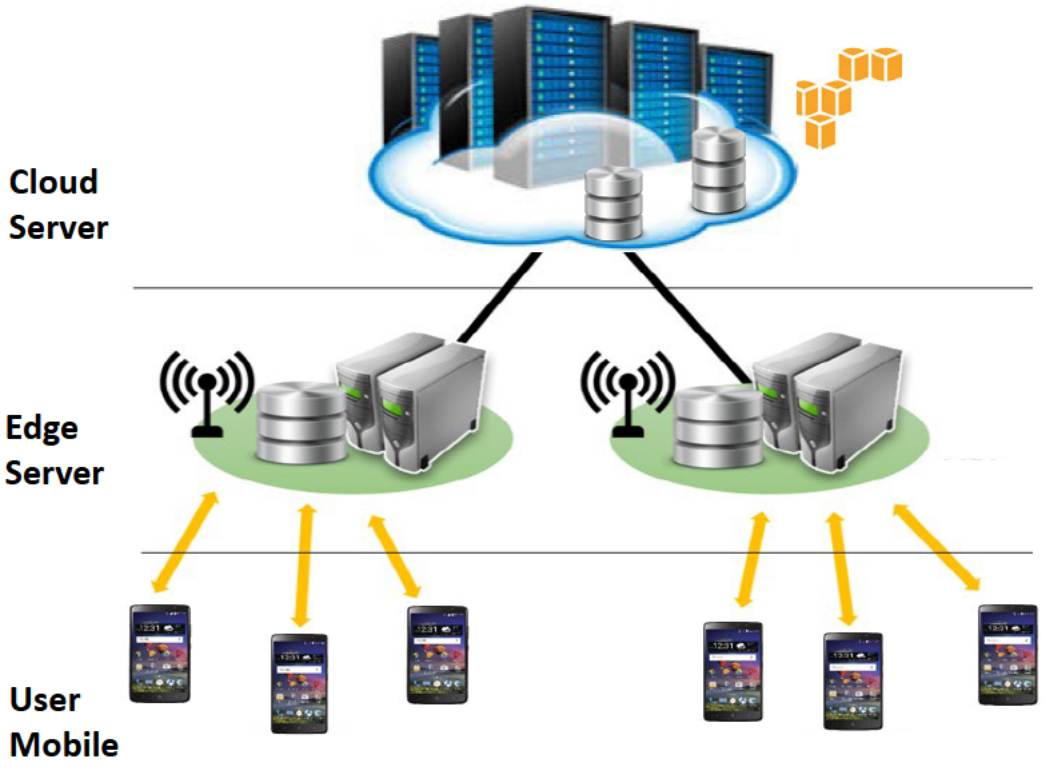


FIGURE 1.2: Mobile-edge cloud computing general architecture

Application partitioning is the process of breaking the application code into units and identifying which units can be processed either locally or in the cloud. Code partitioning occurs at the code level, and it needs to be analysed to define the most computation or storage-expensive operations to be migrated to cloud resources. Remote execution approach indicates the use of web services as application interfaces with computation resources such as private clouds, public clouds or ad-hoc mobile clouds [47]. The main concept of service-based offloading is that platform-independent mobile cloud applications can be developed independent of certain type of mobile devices or operating systems [48]. This allows high utilization of computation resources accessibility and availability. Kumar et al. [44] provided a high-level architecture for a computation offloading system. The offloading monitor collects real-time data from connected mobile devices and mobile networks. Generally, the volume of collected data determines when the offloading planner makes the offloading decision (locally or remotely). Finally, the offloading engine executes application tasks based on the planner's decision. Multiple parameters

control the offloading decision. The computation size and the amount of data to be sent in the communication have priority [44].

1.3 Problem Statement and Research Questions

Mobile offloading is a process of migrating computation-intensive tasks to powerful resources to overcome limitations of mobile devices, notably low processing capability and short battery life. However, not all applications are only computation-centric; some, such as customised data analytic services, natural language processing and face recognition, involve handling huge amounts of data, particularly those interacting with enormous numbers of users. An example of this scenario is the high and active usage of intelligent assistant applications such as Apple Siri [49] and Microsoft Cortana [50] to send huge numbers of service requests to the cloud. This requires significant amounts of data to be sent to the cloud over the wireless network and puts substantial computational pressure on the data centre [4]. Wang et al. [18] argued that data-intensive application computations on mobile computing are always costly in terms of computation time, mobile energy and resource cost. Nan et al. [51] studied the challenges of data-intensive aware mobile applications, and highlighted the significance of increasing data size on monetary cost and QoS improvement.

1.3.1 Motivation

Data-intensive applications bring additional challenges for energy and cost optimisation [18]. Processing large data files on mobile devices has direct overheads in terms of device energy, whereas transferring large data files over mobile networks can increase the device idle time, as well the total computation cost and time. Moreover, data-intensive applications are resource-hungry and need computation machines equipped with powerful CPUs and huge memory to load dynamic application code and data.

In the context of data-intensive mobile applications, an offloading system should be capable of resolving the issues of processing and transferring large chunks of data over heterogeneous networking systems [22]. Thus, it is essential to study the role of data size on application offloading in terms of modelling, planning and optimisation. Franzago et al. [52] defined data-intensive mobile applications as "applications whose primary purpose is to present a large amount of content to a variety of possible users". In addition, data-intensive mobile applications present huge amounts of data to a variety of mobile users [52], with characteristics of these applications being the need for a high-performance computing environment and the abstraction of the computing platforms. Data-insensitive application offloading issues are illustrated below.

1. *High latency and low bandwidth.* Basically, data-intensive application offloading determines whether large data files are transferred in batches or streams. The network performance significantly affects the offloading process. Data transmission with low-bandwidth network channels and high latency is an obstacle to the achievement of offloading system objectives at low cost and reduced energy consumption [24, 53, 54]. An offloading system should generate an optimised task scheduling plan to reduce large data migration when poor network conditions are existed.

2. *Offloading monetary cost.* Offloading mobile applications requires the use of a variety of resources for data transfer and application processing purposes. Offloading cost includes the costs of remote computation at edge or cloud layers, data storage, and transferring data over communication networks. To date, only a few researchers have attempted to estimate these costs for data-intensive applications; the role of data volume needs to be studied carefully to understand its effect on application offloading optimisation.

3. *Short battery life.* Limited device energy is the fundamental issue of most mobile application offloading frameworks [55]. However, in the context of offloading data-intensive applications, this issue can be even more challenging

due to the requirements of migrating large chunks of data over unstable network channels.

4. *Offloading decision complexity.* Decision-making is the core and the most complex part of any offloading framework [56]. The decision process is required to produce a cost-efficient offloading plan to map application tasks to available resources. The offloading plan is generated either statically or dynamically. The former is simpler and mostly applied to predictable workloads, while the latter can be complicated with large-scale problems and in unstable execution environment contexts [57]. An offloading system should consider the application structure, data dependency between tasks and data locations.

To meet the challenges outlined above, the main motivation of this thesis is to design and develop offloading techniques and algorithms to improve the efficiency of data-intensive mobile applications offloading and scheduling on various computing systems. The next section provides the main research questions which this thesis attempts to answer.

1.3.2 Research Questions

The research questions that this thesis seeks to answer, related to mobile data-intensive applications offloading and scheduling, can be expressed as:

1. How is the modelling of mobile data-intensive applications different from the ones of other application types such as computation-intensive and bandwidth-intensive applications?

2. Which optimisation techniques can meet the challenges of constructing data-intensive application offloading plans in terms of application complexity, data variation and mobile network instability?

3. How does the problem scale and application complexity of scheduling data intensive mobile applications influence the selection of an optimisation technique?

4. What are the determinants of selecting a mobile-aware computing system, and how are they related to application model and user QoS constraints?

1.4 Contributions

Data-intensive mobile application execution on an MECC system involves proposing efficient offloading techniques and algorithms to conquer the physical limitations of mobile devices, as well as meeting user QoS constraints associated with contextual and networking status. This thesis proposes techniques and algorithms to plan the execution of data-intensive mobile applications in mobile-aware computing environments. Allocating application tasks to computation nodes (local, edge, or cloud) is formulated as an optimisation problem, which aims for the minimum of energy and cost under the constants of available energy and task deadline. The following are the key contributions of this thesis.

- The identification and description of the challenges of data-intensive application offloading on multiple computing systems. The data size parameter needs to be considered from two perspectives: how the increase in data size impacts the computation complexity of an application and accordingly the offloading technique complexity, and the data communication overhead of uncertain mobile network quality and bandwidth. This thesis provides a data-oriented mobile application modelling solution, which is clearly stated in a cost model formulation to reflect the role of data variation in offloading planning and optimisation.

- A resource allocation algorithm that leverages particle swarm optimisation (PSO) to generate an application offloading plan on a hybrid MCC system. PSO is adopted for its efficiency in searching the solution space for a global

and optimal offloading plan. This thesis demonstrates the viability of adopting an evolutionary search optimisation with PSO, which is not commonly applied in the literature.

- A resource allocation and task scheduling algorithm which is based on mixed integer linear programming (MILP) technique for data-intensive application on MECC. The MILP technique is convenient for large-scale search spaces. This thesis is unique in proposing linear search optimisation for large-scale mobile applications, which involves handling the complexity of managing data and computation distribution in a multi-layer computing system.

- An experimental analysis of various data-intensive application offloading setups/scenarios in regard to application models, computing systems and model parameters. The analysis provides a detailed investigation of how these parameters influence the application decision, as well as related optimisation variables of time, energy and cost.

1.5 Thesis Organisation

This thesis includes seven chapters, which are organised as shown in Figure 1.3. An overview of the thesis's organisation is given below.

- Chapter 2 provides a comprehensive survey of the research literature on resource allocation and task scheduling algorithms and techniques for computation and data-intensive applications on mobile-aware computing systems.

- Chapter 3 presents a QoS-constrained resource allocation algorithm for data-intensive applications on a hybrid mobile cloud computing system. PSO was adopted due to its ability to find a global optimum solution from a solution search space (i.e., application offloading plans). This chapter is published in the following conference paper:

  – Alkhalaileh, Mohammad, Rodrigo N. Calheiros, Quang Vinh Nguyen, and Bahman Javadi. "Dynamic resource allocation in hybrid mobile

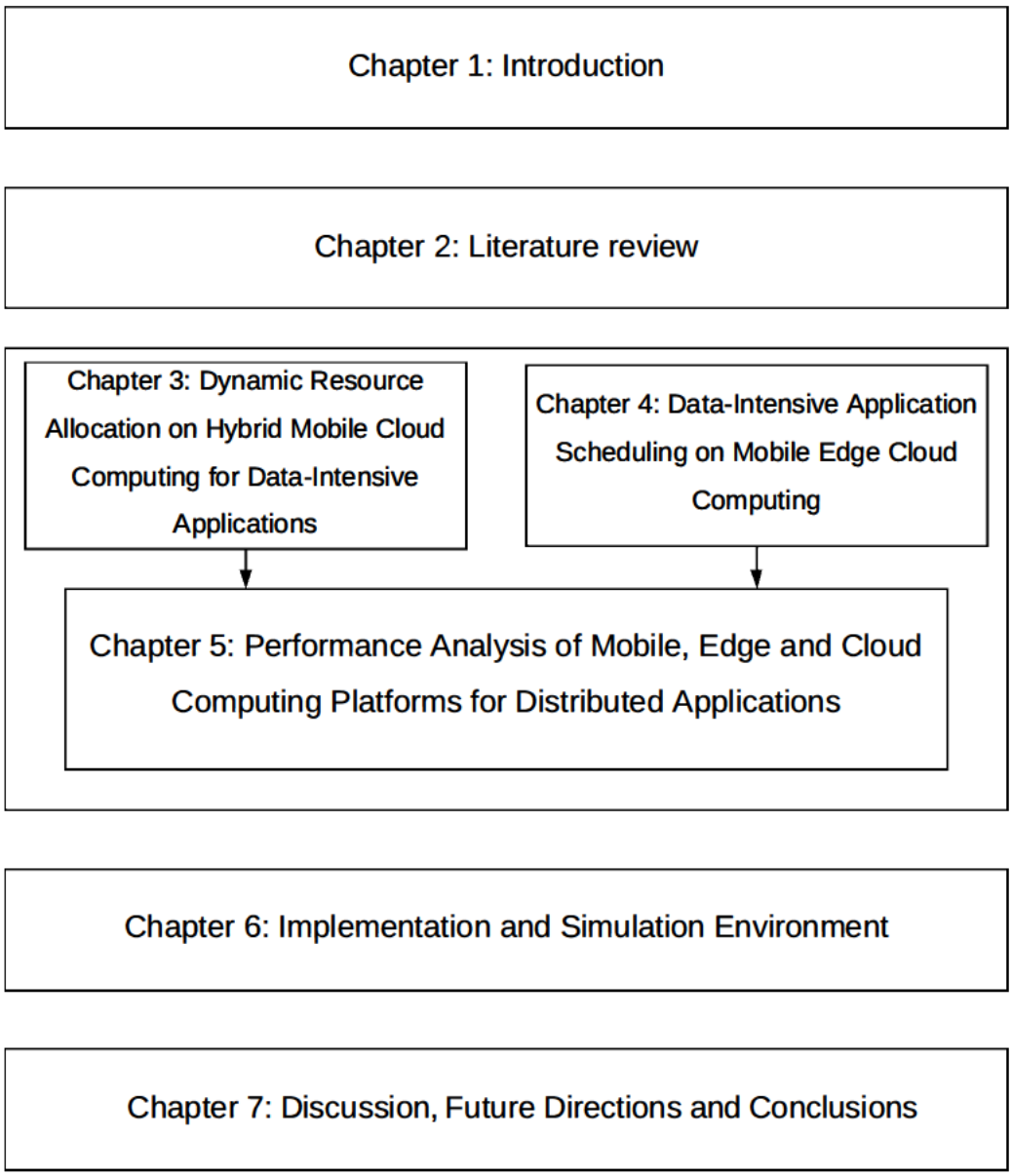


FIGURE 1.3: Thesis organisation

cloud computing for data-intensive applications." *In International Conference on Green, Pervasive, and Cloud Computing, pp. 176-191. Springer, Cham, 2019.*

- Chapter 4 proposes a scheduling algorithm for a large-scale data-intensive application on an MECC. To overcome the time required for the PSO technique to schedule complex applications, a MILP technique was adopted to

perform linear search optimisation under constraints of mobile energy and task deadline. This chapter is published in the following journal paper:

  - Alkhalaileh, Mohammad, Rodrigo N. Calheiros, Quang Vinh Nguyen, and Bahman Javadi. "Data-intensive application scheduling on Mobile Edge Cloud Computing." *Journal of Network and Computer Applications (2020): 102735.*

    MVC

- Chapter 5 provides a comprehensive analysis of many aspects of mobile-aware computation models. The objective is to provide recommendations for a mobile-related computation paradigm to execute mobile applications based on application model, application data size, mobile network performance, and mobile energy status. This chapter is published in the following book chapter:

  - Alkhalaileh, Mohammad, Rodrigo N. Calheiros, Quang Vinh Nguyen, and Bahman Javadi. "Performance Analysis of Mobile, Edge, and Cloud Computing Platforms for Distributed Applications". *In MEC2020, Mobile Edge Computing ("in press, 2020.")*

- Chapter 6 provides a framework for evaluating the ability of the proposed algorithms and techniques to reduce the energy and monetary costs of running data-intensive mobile applications on mobile-aware computing systems. The framework consists of a set of components, comprising services of profiling, resource investigation and communication, context monitoring, an offloading decision-maker, and a QoS optimiser.

- Chapter 7 concludes the thesis, summarises its findings, and suggests directions for future work.

Chapter 2

Literature Review

2.1 Introduction

Mobile Computing has physical limitations on its ability to process complex tasks and handle large data sets. A principal concern related to computation-intensive and data-intensive tasks is power consumption. As noted in Chapter 1, a general term for methods designed to overcome the physical (notably, power) limitations of mobile devices is "computation offloading", which refers to the migration of task complexity from limited resources to more powerful resources. For example, integrating cloud computing infrastructure with mobile computing may solve the current technical and physical limitations of the latter [18]. Satyanarayanan et al. [24] argued that cloud computing is the most convenient computing architecture to overcome the shortcomings of mobile computing systems. The evolution of cloud computing grants the capability to resolve many issues of mobile computing, due the unlimited resources it offers in high scalability and reliability [58]. MCC, a combination of high-performance clouds and convenient mobility-enabled devices, is emerging as a convenient computing model to augment the computational abilities of mobile devices through the extraordinary computation facilities of the cloud.

Computation offloading can improve performance in data-intensive mobile applications by migrating intensive computations from resource-limited devices (mobile devices) to resource-rich machines such as cloud or nearby resources (servers) to minimise execution time and optimise device energy [5–7]. This chapter discusses research intended to overcome the challenge of designing and implementing convenient offloading decision-making techniques with respect to various offloading quality measurements, including energy consumption, latency and responsiveness, and monetary cost. However, there are undeniable challenges related to the offloading process. The next section focuses on the main research work undertaken in attempts to resolve the aforementioned offloading challenges.

2.2 Computation Offloading Techniques and Algorithms

Computation offloading has been studied intensively, and several techniques have been proposed for optimising energy consumption and meeting user QoS metrics such as response time and monetary cost. These techniques include computation augmentation [21], device cloning [59], nearby-resource computation [24], provisioning middleware [23, 60, 61] and context-aware and profiling MCC system [25, 62].

Mobile cloud computing has been widely accepted as a computation architecture to overcome mobile computing's shortcomings of limited computation capacity and short battery life. As noted earlier, MCC involves the integration of mobile devices and public cloud resources to provide computation and storage services for mobile applications in a scalable manner, while providing high QoS [63]. However, MCC has two significant limitations: long propagation distance from mobile device to a remote cloud centre, which results in an excessively long latency for mobile applications, and poor usage of computation and storage at the network edges, which would otherwise be sufficient to enable ubiquitous mobile computing. However, the research literature shows alternative computing models can handle the

latency and resource utilisation issues, such as MEC [40]. Processing on close edge nodes is more efficient than cloud or cloudlet for reducing data transfer latency and capturing more real-time context information [34].

Several definitions have been proposed for edge computing; in most, edge computing is the representation of an intermediate resource layer, including, micro data centres, edge devices and network resources, which are able to produce, collect and process data as well as migrate data to the cloud. Edge computing definitions cover frameworks, applications, technologies and capabilities [64]. Tseng, Canaran and Canaran [65] provided a definition that is closely related to computation offloading scope:

"Edge computing, simply known as Edge, brings processing close to the data source, and it does not need to be sent to a remote Cloud or other centralised systems for processing. By eliminating the distance and time it takes to send data to centralised sources,the speed and performance of data transmission improves, as well as the devices and applications on the Edge."

Mobile edge computing is an emerging paradigm designed to meet the ever-increasing computation demands of mobile applications [66]. Most studies on offloading optimisation in MEC focus on finding an optimal strategy for distributing application tasks to edge resources, attempting to minimise data transfer latency with respect to the capacity of edge nodes [29, 34, 40, 67–70]. The main feature of MEC is to push mobile computing, network control and storage from resource-limited mobile devices to the network edges to enable computation-intensive and latency-critical applications [35]. With MEC, a mobile application can track real-time information such as behaviours, location and environment, which reduces the exchange of sensitive information between the mobile device and cloud resources, while being more energy efficient. However, edge resources are limited in computation capacity, scalability and high-energy sensitivity to perform long-term computation. One solution is integrating edge and cloud resources, combining benefits from the high capability of cloud resources and the availability of access resources at the

edge layer. As noted earlier, this joint computation architecture is termed mobile-edge cloud computing [41]. MECC differs from traditional cloud computing in combining cloud and mobile computing models via wireless networks, and has the advantage of allowing cloud resources to handle computation and data-intensive tasks [18].

The rest of this section presents a comprehensive survey of the literature on computation offloading optimisation with consideration of issues like energy consumption, resource scalability, application responsiveness and data sensitivity.

2.2.1 Energy Consumption

It is critical for mobile applications to deliver complex high-performance functionalities at low cost. Mobile device input/output (I/O) processing and network communications are energy-hungry components [21]. Abolfazlia et al. [21] showed that mobile computation augmentation not only delivers intensive computation capacity to mobile users, but can save energy use and prolong battery life. Offloading heavy tasks to the cloud can reduce energy consumption efficiently. Thus, energy consumption is a fundamental objective while implementing offloading techniques [71].

Chun et al. [59] designed the CloneCloud MCC model to bring the power of cloud computing to mobile smartphones. A clone is a mobile image hosted on the cloud. They observed that application code partitioning was not always viable due to it being hard to split correctly in respect to connectivity. In addition, CloneCloud only considers limited input/environmental conditions in offline pre-processing and needs to be bootstrapped for every new application built [23]. Moreover, in CloneCloud the virtualisation overhead and the recurring tasks of synchronising the shared data between the mobile and cloud degrade the augmentation performance [72]. Another disadvantage is the necessity to access device services such as global positioning systems (GPS) and Bluetooth. Transferring device I/O between the mobile and the clone environment over the network also

reduces the responsiveness and increases power consumption [73]. Furthermore, running an application on multiple platforms increases the complexity of device management [73].

Cuervo et al. [74] argued that the critical obstacle for future growth of smartphone use is battery energy. They proposed MAUI architecture to address both energy and performance for code-offloading MCC applications. MAUI code offloading depends on remote execution to take advantage of the resource-rich infrastructure of remote servers. There are two remote execution options. The first relies on the programmer to pinpoint remote partitions and how to adopt the partitioning schema to the changing network connection [75]. The second is to use full virtual machines (VM) migration for individual applications. MAUI builds proxies, profilers and solvers on both the client and server sides. The model benefits from remote execution by maximising energy saving through a fine-grained code offload, while minimising the changes required to applications. One disadvantage of MAUI is that not all code categories can be offloaded because constraints on some application program interface (API), I/O devices or internal resources on mobile devices have been established. A further disadvantage is that the decision process in MAUI only considers information that is coarse-grained, compared with the complex characteristics of the mobile environment, and MAUI is limited to work on scalable systems on the cloud [23].

While CloneCloud [59] and MAUI [74] emphasise reducing energy consumption by decomposing an application, they do not consider the reusability of computation functionalities across applications [22]. Kemp et al. [76] introduced a programming model, Cuckoo, which is designed for Android. The model supports both local and remote service executions via user friendly interfaces to bundle and run application packages. The offloading decision is based on network bandwidth and signal quality, and it uses heuristic data and context information to evaluate an offloading decision.

March et al. [22] designed μCloud architecture with the aim of developing rich mobile applications that minimise energy cost through application decomposition

and adopting component reusability. The application is defined as a cyclic graph from a set of heterogeneous components, and each component represents a high-level functionality. Components can be native mobile, cloud, or a combination of both. The offloading framework consists of application and execution models. The application model manages the coordination between application tasks in terms of passing data, shared execution space and virtual memory management. The execution model partitions the application graph into disjoint sets of partitions, where each represents the execution of a component in a target execution node, mobile or cloud. The main advantage of μCloud is the ability to have seamless interaction between execution components to reduce data migration and thus communication energy.

Ding et al. [77] proposed a VD-ABC technique, a metaheuristic technique based on bee colony optimisation and Boltzmann selection strategy to handle cloud VM placement complexity with a multi-user offloading schema. The technique improved CloneCloud [59] by reducing the complexity of multi-users offloading by determining communication and waiting energy costs. Similarly, Kao et al. [78] extended CloneCloud [59] for collaborative task execution between a mobile device and its cloud clone to execute deadline-constrained applications. The task scheduling problem was formulated as the shortest-path problem on a directed cyclic graph (DAG) model. EnaCloud [79] worked more restrictively and proposed a heuristics approach for dynamic VM placement in the remote cloud as a bin packing problem. However, EnaCloud only considered energy consumption overhead and disregarded communication and data transfer costs.

Terefe et al. [80] proposed an energy-efficient offloading approach based on a discrete time Markov chain to model data communication channels between mobile devices and cloud servers. The authors highlighted the concept of multi-server offloading to benefit from the ability of mobile devices to access multiple cloud providers. This offloading distribution can ensure high QoS achievement as well as more realistic networking data profiling than a single site offloading execution. The approach aims to find the optimised partitioning plan to reduce energy while

meeting a task execution deadline. The work simulates the offloading decision-making for linear-based workflows and consider workflow tasks as separate execution components at different granularity levels. Experimental results showed that the multisite computation offloading approach provided savings compared to single-site execution with respect to both energy consumption and execution time.

Goudarzia et al. [69] applied the same partitioning strategy to hybrid multi-site offloading to reduce energy consumption. To address the challenges of single-site offloading, a fast hybrid multi-site computation offloading solution. The solution handles application complexity for offloading optimisation and proposed methods for optimal and near-optimal offloading partitioning. The offloading process is adaptive in selecting the optimisation technique, such that the branch and bound algorithm (BB) is applied to accelerate search space through a suitable bounding function. For a near-optimal method, PSO is applied to reduce the polynomial time for large-scale problem optimisation.

Chen et al. [81] studied the multi-user computation offloading problem for MECC in a multi-channel wireless interference environment. The authors argued that MCC can generate a critical latency issue while exchanging data via a wide communication network. Moreover, cloudlet-based models can have some limitations due to the limited converge of WiFi networks and the limited support of utilising high powerful resources, which may not align with QoS achievement for a large number of users. To resolve these issues, Chen et al. adopted mobile-edge computing with integration with public clouds to allow energy-efficient offloading distribution to reduce execution time and cost. They aimed to select offloading that reduced energy consumption and data transmission latency. The optimisation problem was formulated as an offloading game to allocate tasks to edge servers. Results show that the proposed game-based technique is able to achieve good offloading performance even with large-scale problems.

Most of the energy-based MCC solutions follow the approach of mobile augmentation through extending cloud services into mobile devices, and enabling mobile devices to collaborate with cloud resources [82]. However, these solutions mostly

do not consider application complexity variation and context parameters, such as input data size, network bandwidth and corresponding data communication costs. The mobile/wireless network performance significantly affects mobile application responsiveness, processing time and energy consumption [15]. Satyanarayanan et al. [24] proposed an architecture to improve application responsiveness and availability by relying on high-performance cloudlets, which have good connectivity with cloud servers. However, cloudlets are limited in their convergence on the mobile network for service provisioning, which does not support large numbers of mobile users to share available resources [81] .

Hu et al. [83] studied the effects of adopting cloudlets on offloading systems in terms of energy consumption and latency. Experimental results showed that offloading to cloud servers can reduce performance and increase energy consumption compared to running locally on a mobile device. For highly interactive applications, offloading to nearby resources like cloudlets and edge resources can improve offloading performance. Zhang et al. [84] studied deadline-constrained offloading of a sequence of tasks in a collaborative MCC environment. The offloading optimisation problem was formulated as a constrained shortest-path problem. The study showed that collaborative task execution with consideration of task dependencies can significantly reduce energy consumption in comparison to local and remote execution.

Cardellini et al. [85] designed a game-theoretic approach to compute offloading in MEC and used the waiting time in computation nodes to separate the execution of multi-user applications. These researchers proposed efficient techniques for adopting cloudlet/edge computing to enhance mobile application responsiveness, reduce latency and optimise device energy. However, they did not attempt to integrate between MCC and MEC, and in particular they did not propose practical and adequate resource allocation and task scheduling techniques in the context of data-intensive mobile applications. Moreover, the techniques they discussed do not relate to offloading decision optimisation behaviour based on data-aware parameters, such as application data size and location.

Jararweh et al. [68] supported the argument that energy consumption optimisation is the central issue of computation offloading and mobile augmentation. The authors proposed an energy-aware optimisation technique based on MILP in an edge system to reduce delays in task execution. Jinke et al. [41] attempted to identify the system configuration with least latency by comparing mobile-aware computing systems using Lagrange optimisation. They found that collaboration between cloud and edge computing can efficiently reduce end-to-end computation latency by reducing device communication and waiting energy.

Wang et al. [86] extended the MEC computing model by taking advantage of a central cloud to overcome the physical limitations of edge clouds. However, the proposed offloading strategy gives the highest priority to the central cloud as the offloading target, and local task allocation is only determined when no connection between mobile device and the central cloud is available.

The research discussed here proposed efficient techniques for adopting edge computing to enhance mobile application responsiveness, reduce latency and optimise device energy. However, previous researchers have done little to integrate MCC and MEC, particularly in the context of data-intensive mobile applications. In this thesis, interaction was extended to involve an intermediate layer of resources and by considering parameters of cost and task deadline.

2.2.2 Elasticity and Scalability

Resource elasticity and scalability are critical factors for data-intensive application offloading. Scheduling resource allocation is also a crucial issue for achieving scalability and elasticity. The resource allocation may include the CPU, memory, storage (disk I/O), and network bandwidth resources [18].

ThinkAir [23] is a framework for code offloading that enables mobile application developers to scale cloud resources with parallel execution of offloaded tasks. The proposed architecture framework is similar to CloneCloud [59], in which computation offloading is handled in a VM image inside the cloud. The main difference

between the ThinkAir framework and CloneCloud is that it is more sensitive to resource scalability and elasticity by enabling VM image sharing on the cloud. Decisions about computation offloading resources are based on the execution time, energy consumption, and heuristics information about previous executions. The framework combines three main modules for application execution, resource management and data profiling. ThinkAir supports resource auto-scaling to reserve load balancing as well as application performance at cloud server level. Moreover, the proposed framework relies on different profiling levels for improved offloading performance and energy consumption estimation. At hardware level, only data about device resources is profiled, whereas at software level, many parameters are monitored, including program execution, network state and amount of transferred data within an interval of time. For network performance measurement, a network profiler collects low-level details about the usage of different network interfaces, including data transmission rate, number of transmitted packets and network propagation delay. In contrast, this thesis provides more broad profiling services and covers both local and remote resource monitoring.

Zhang et al. [73] assumed that augmenting mobile applications to the cloud can benefit from the scalability and elasticity of cloud resources. They proposed a middleware to deploy elastic applications that consists of multiple components called weblets. A weblet can be executed or launched either on a mobile device or in the cloud environment. Weblets may improve offloading robustness through adopting parallel execution schemes or algorithms of intensive computing components [21].

Rahimi et al. [60] proposed MAPCloud, a QoS-based two-tier resource-based MCC. The work mainly targets system scalability and performance. Their model deals with a mobile application as a workflow of tasks in which the workflow execution is apportioned and mapped to computation resources based on average optimisation of service price, latency, and power. Later, the model was extended as MuSIC (Mobility-Aware Service Allocation on Cloud) [87], which uses mobility information to estimate and predict the optimised resource allocation and execution resources.

Context-aware offloading models emphasise profiling network conditions for offloading decision prediction, but none of their authors have studied offloading decision optimisation behaviour based on data-aware parameters, such as application data file size and location. On one hand, the processing of huge data files can drastically absorb device energy. On the other hand, transferring these large files can increase the device idle time and thus consumes more waiting energy for task completions. In addition, service availability and performance can critically affect the amount of waiting energy for remote execution services [88].

Smartphones now include tools and technologies, such as sensors and GPS, to discover and learn about the surrounding environment and agents. For example, location-based applications can be provided from mobile devices to allow tracking of agents and items and enhance the performance of companies who have employees deployed in the field. La et al. [89] developed a framework for location-based services to monitor mobile user context information. The acquired information is transmitted to the cloud for analysis, and hence, the user location can be predicted.

Zhou et al. [90] proposed a context-aware MCC model (mCloud) that benefits from the changing context of a mobile device and heterogeneous cloud resources for providing an adaptive and seamless offloading decision-making capability. The work objective is to enhance service availability and performance by proposing multi-layered offloading destinations (local device, cloudlet, mobile ad hoc and public cloud) with various wireless communication options (Bluetooth, WiFi, 3G and 4G).

Lin et al. [62] proposed a context-aware decision making system for execution offloading. They argued for the significance of embedding context information, including network robustness, network congestion, and time of day, in anticipating the offloading decision. They integrated their model with the ThinkAir [23] architecture. Experimental results demonstrate high percentage accuracy and convenient offloading performance in terms of the response time and energy saving.

Chang et al. [91] presented an edge cloud system to extend the data cloud for workload distribution for all the path between the user device and the cloud. The model

is a collaborative system between edges and cloud computing systems for running low-latency and bandwidth-sensitive applications. Edge nodes are grouped based on the zone location. Result demonstrate the capability of edge cloud localisation offloading to save energy and scale real-time services. Souza et al. [92] worked on latency-sensitive optimisation and adopted fog cloud scenarios for lightweight and resource-intensive services. The authors formulated the allocation optimisation as an integer linear programming (ILP) problem, and showed that fog cloud collaboration reduces the number of requests to the cloud and enables low delay for the low-level and medium-level tasks complexity. Kang et al. [4] also worked on collaborative edge cloud system. The authors proposed a fine-grained latency-sensitive offloading technique with deep a neural network to improve end-to-end latency while minimising overall energy consumption on user's devices. Results demonstrate significant energy savings and greater system utilisation, 59.5% and 1.5x respectively.

2.2.3 Data-intensive mobile applications

As mentioned previously, data-intensive mobile applications enable the processing and the migration of huge amounts of data to a variety of mobile users [52]. Therefore, data-intensive application offloading must be studied with respect to the impact of data size, and corresponding parameters such as network bandwidth and application complexity.

Data-intensive applications, such as customised data analytic services, natural language processing and face recognition, are resource-intensive applications that require machines with powerful CPU and huge memory space to load dynamic application code and data. Nan et al. [51] studied the challenges of increasing user quality of experience (QoE) when running mobile data-intensive applications. The study revealed strong associations between mobile users, QoE and mobile application responsiveness, and further, showed how cloud services can minimise

application response time, thereby, improving the overall mobile application performance. Moreover, the study used monetary cost optimisation to improve user QoE significantly.

Cloud-based resource allocation for mobile data-intensive applications is challenging. Simultaneously minimising monetary costs and enhancing customers' QoE requires an efficient cost-optimisation model [93]. Kang et al. [4] proposed a data-centric offloading framework called Neurosurgeon, which differs from a control-centric framework because it produces execution plans or partition decisions according to the structure of data topology and data dependency between application tasks. The work provides comparisons of most relevant offloading techniques and includes the contribution of application data size to data transfer overhead, application partitioning, execution time and profiling. The results show some interesting insights related to experimental work with respect to variations of network types, bandwidth capacity, and computation power. Firstly, for data-intensive applications, the data transfer latency is often higher than mobile computation latency, particularly with 3G/4G networks. Secondly, even though cloud processing is significantly more powerful than mobile processing, for applications that involves intensive data communication, offloading solutions need to illiterate an efficient edge cloud collaboration to settle. Lastly, local processing often incurs less latency and energy consumption than cloud-only processing. Cloud-only can achieve higher performance when a WiFi connection is available.

The general schema of mobile offloading is often preferable with computation-intensive tasks where the amount of data communication is minimised [94]. This does not support the requirements of real-time and data-intensive application. Thus, a hybrid MCC architecture can be proposed to meet the requirements of data-intensive mobile applications.

Zhou et al. [90] proposed a three-tier MCC middleware that empowers programmers with computation alternatives, based on the application cost model and the offloading decision-maker. Even though the proposed model includes the task data size in generating an optimised execution application tasks plan, the data size is

marginally small and cannot reflect the scenario of data-intensive application tasks scheduling on MCC. In addition to offloading data size, the selection of proper application and programming models is another challenge for accurate estimation of energy consumption and efficient management of simultaneous offloading by multiple users [95].

Processing on close edge nodes is efficient for reducing data transfer latency and capturing real-time context information [34]. In the context of data-intensive mobile applications, one can observe certain issues related to MECC. Firstly, transferring huge amounts of data via an unstable mobile network will lead to an unpredictable increase in data transfer latency. Secondly, edge resources would have limited capability to process application tasks with high data input. Finally, transferring huge data volumes over the cellular network and processing in public clouds demands new techniques for data-intensive optimisation to reduce energy consumption and monetary cost.

For joint distribution offloading on MECC, some researchers have proposed offloading techniques for latency-sensitive applications and to achieve user QoS requirements. Such techniques support parallel computation on accessible resources on MECC. Zhao et al. [96] designed a threshold-based policy to improve the QoS of MEC, involving combination of local edge resources with public cloud resources. Enzai et al. [67] developed a heuristic algorithm for multi-site computation offloading in MECC under multiple objective optimisations of energy, time and cost, using a heuristic approach to transform the multi-weight optimisation to single-weight on workflow applications. Vu et al. [97] proposed a joint task offloading and resource allocation optimisation problem, and recommended applying a MINLP technique to minimise the mobile energy consumption under the fog nodes, resource constraints and task delay requirements.

Multi-user edge computing can improve MEC by providing efficient management and fair distribution of edge resources. Zhao et al. [96] designed a threshold-based policy to improve the QoS of MEC, involving the cooperation of local edge resources and public cloud resources, which takes advantage of the low latency

of the local cloud and abundant computational resources of the public clouds simultaneously. Enzai et al. [67] developed a heuristic algorithm for multi-site computation offloading in MECC and considered multiple objective optimisations of time, cost and energy. The idea was to transform the multi-weight optimisation to single-weight for a workflow application using a heuristic approach. Technique evaluation results have shown that the heuristic algorithm can produce good quality solutions in a reasonable time for those test problems. Vu et al. [97] proposed a joint task offloading and resource allocation optimisation problem based on mixed integer nonlinear programming (MINLP) technique to minimise the energy consumption for mobile devices under the fog nodes, resource constraints and task delay requirements. The experimental results demonstrated the technique's ability to reduce energy consumption while achieving all deadline constraints.

The literature includes many studies of ways to meet the requirements of computation offloading. Some of these focus on techniques to coordinate computing resources, while others pay more attention to offloading optimisation techniques and algorithms. However, few researchers have attempted to map data-intensive offloading challenges and proposed data-centric offloading and scheduling frameworks and techniques. Table 2.1 summarises the main work to date on computation offloading and device augmentation on mobile-aware computing models(MCC, MEC and MECC).

TABLE 2.1: MCC models summary

MCC Model	Optimization Metrics	QoS Metrics	Offloading Technique	Resource Allocation Algorithm / Technique	Computation Resources	Application Type	Evaluation Method
CloneCloud [59]	Device energy	Execution time	Code offloading	History based profiling ILP	Public cloud Nearby infrastructure	Computation intensive	Experiments
Cloudlet [24]	Responsiveness	Response time	VM migration	VM migration and application profiling	Nearby infrastructure	Computation intensive	Experiments
MAUI [74]	Device energy	Execution time	Remote execution	History-based profiling 01 ILP	Public cloud Nearby infrastructure	Computation intensive	Experiments
WEBLET [73]	Device energy Monetary cost	Context awareness Application throughput Response time	Remote execution	Application partitioning	Public cloud Scalability enabled	Data intensive	Experiments
μCloud [22]	Device energy	Security Usability	Remote execution	Graph based decomposition	Public cloud	Computation intensive	Experiments
ThinkAir [23]	User-defined	Robustness	Code offloading	History-based profiling VM parallelization	Public Cloud Scalability enabled	Computation intensive	Experiments
SAMI [98]	Responsiveness	Portability Trust	Remote execution	Application profiling	Public cloud Mobile operator	Computation intensive Data intensive	N/A
MAPCloud [60]	User-defined	Robustness	Remote execution	Graph based decomposition	Hybrid cloud Scalability enabled	Computation intensive	Simulation

Table 2.1 continued from previous page

MCC Model	Optimization Metrics	QoS Metrics	Offloading Technique	Resource Allocation Algorithm / Technique	Computation Resources	Application Type	Evaluation Method
HMCC [99]	Device energy Responsiveness	Response time	Remote execution	Application partitioning	Public cloud	Computation intensive	Experiments
Zhou et al. [90]	Device energy Monetary cost	Context awareness	Code offloading	Application profiling	Public cloud Nearby infrastructure Ad-hoc mobile cloud Scalability enabled	Computation intensive	Experiments
Giurgiu et al. [61]	User-defined	Response time	Code offloading	Graph based decomposition	Public cloud	Computation intensive	Experiments
Lin et al. [62]	Device energy Responsiveness	Context awareness Accuracy	Code offloading	Application profiling	Public cloud	Computation intensive	Experiments
Enzai et al. [67]	Monetary cost Device energy	Response time	Computation offloading	Greedy Algorithm Workflow partitioning	Public cloud	Computation intensive	Experiments
Jinke et al. [41]	Responsiveness	N/A	Computation offloading	Application partitioning Non-liner optimisation	Public cloud Edge computing	Computation intensive Data intensive	Simulation
Jararweh et al. [68]	Responsiveness Device energy	SLA	Computation offloading	MILP	Public cloud Edge computing	Computation intensive	Experiments
Goudarzia et al. [69]	Responsiveness Monetary cost Device energy	N/A	Computation offloading	Graph-based decomposition 0-1 ILP	Public cloud Mobile	Computation intensive	Simulation Experiments

Table 2.1 continued from previous page

MCC Model	Optimization Metrics	QoS Metrics	Offloading Technique	Resource Allocation Algorithm / Technique	Computation Resources	Application Type	Evaluation Method
Chen et al. [70]	Responsiveness Device energy	Response time	Computation offloading	0-1 MINLP	Public cloud Edge computing	Computation intensive	Simulation
Chen et al. [81]	Responsiveness Device energy	N/A	Computation offloading	Multi user offloading game	Edge computing Mobile	Computation intensive	Simulation
Cardellini et al. [85]	Responsiveness Monetary cost Device energy	Response time	Computation offloading	Generalized Nash Equilibrium Problem (GNEP)	Public cloud Edge computing Mobile	Computation intensive	Simulation
Wang et al. [86]	Responsiveness Device energy	Efficiency	Computation offloading	MILP	Public cloud Edge computing Mobile	Computation intensive	Simulation
Souza et al. [92]	User-defined	Response time	Computation offloading	0-1 ILP	Public cloud Fog computing	Computation intensive	Experiments
Zhao et al. [96]	Responsiveness	User-defined	Computation offloading	Threshold-based policy	Public cloud Edge computing	Computation intensive	Simulation

2.3 Summary and Discussion

Previous researchers strove to overcome the challenges of data-aware mobile application offloading. However, their models do not consider the contribution of data size variation in association with other optimisation parameters in application offloading decisions. In addition, these models do little to integrate cloud and edge systems to resolve data-intensive application offloading and scheduling optimisation. Moreover, although the literature presents comprehensive research proposing computation-intensive and data-intensive application offloading optimisation from different perspectives, in a variety of computation environments, two issues are not fully addressed. The first is the efficient adoption of MECC to overcome issues of latency-sensitive application when transferring and processing large data files. The second is offloading optimisation in response to advanced scenarios, such as deadline and multi-user models. This thesis describes algorithms and techniques for data-intensive application scheduling and offloading in hybrid MCC and MECC systems.

The next chapter presents the first known work on data-intensive application offloading in hybrid MCC system.

Chapter 3

Dynamic Resource Allocation in Hybrid Mobile Cloud Computing for Data-Intensive Applications

Chapter 2 shows that considerable research has been directed toward computation-intensive mobile application offloading optimisation, but little towards data-intensive offloading optimisation. As noted earlier, emerging data-intensive applications, such as face recognition and natural language processing, impose challenges on mobile cloud computing platforms because of high bandwidth cost and data location issues. To overcome these challenges, this chapter, proposes a dynamic resource allocation model to schedule data-intensive applications in an integrated computation resource environment composed of mobile devices, cloudlets and public cloud, which can be referred to as hybrid mobile cloud computing.

The allocation process is based on a system model that takes into account parameters related to application structure, data size and network configuration. This chapter describes real experiments conducted on the implemented system and an evaluation of the performance of the proposed technique. Results demonstrate the ability of the technique to generate an adaptive resource allocation in response

to variation in application data size and network bandwidth. The proposed technique improves the execution time for data-intensive applications by an average of 78%, and reduces mobile energy consumption by an average of 87% compared to a mobile device alone, while monetary cost increases only 11% due to using cloud resources and mobile communication.

3.1 Introduction

The use of mobile devices such as smartphones and tablets has increased tremendously due to advancement in functionalities supported by high connectivity, faster CPU, large memory, and sophisticated sensors. The use of mobile technologies (mobile hardware, software, and communication) to create, process and store information without consideration of geographic location is generically described as mobile computing [2]. A common technique to reduce computation overhead on mobile devices is offloading complex tasks to high-capability resources such as clouds [7] and cloudlets [24]. Section 1.2 provides a detailed description of computation offloading and device augmentation concepts.

Cloud computing offers highly capable computation resources in a convenient "pay-as-you-go" cost model [17], and employing cloud resources in computation offloading facilitates MCC. MCC aims to augment the capabilities of mobile devices by improving and optimising their computing performance while undertaking compute-intensive tasks in cloud-based resources [25]. Section 2.2 presents an extensive review of the literature on offloading techniques and algorithms with respect to MCC and hybrid MCC environments. The review concluded that it is critical to find ways to reduce the energy consumption of mobile applications when delivering complex high-performance functionalities [21].

Offloading heavy tasks to the cloud can reduce energy consumption in an efficient way. Energy-based models of MCC proposed in the literature include mobile device cloning in remote resources [59], code offloading and migration [74], application and network profiling [76], and application decomposition and reusability [22, 61]. For

hybrid MCC, cloudlets are integrated with an MCC model to enhance application responsiveness and reduce energy consumption associated with local processing and cloud communication instability [24]. In addition, Zhou et al. [90] proposed hybrid MCC middleware to support application offloading and task scheduling based on a variety of cost models.

However, while some applications are only computation-centric, others, such as customised data analytic services, natural language processing and face recognition, involve handling huge amounts of data, particularly those interacting with enormous numbers of users. The discussion in Section 2.2.3 illustrates the challenges of data-intensive offloading in terms of energy consumption, resource management, application responsiveness and the cost of data communication over cellular networks. To facilitate data-intensive mobile applications, an offloading system should be capable of resolving the issues of processing and transferring large chunks of data over heterogeneous networking systems [22]. Thus, it is essential to study the role of data size on application offloading in terms of modelling, planning and optimisation.

Research on MCC-based offloading to date has not resulted in a clear view of how to handle data-intensive applications and how variation in application data can affect offloading decisions and task scheduling in MCC-based environments. In this chapter, a dynamic resource allocation model to schedule data-intensive applications in a hybrid MCC environment is proposed. A hybrid MCC model is a combination of three type of resources: mobile devices, cloudlets, and the cloud. In addition, a mobile application is defined as a set of independent tasks and handled as a bag of task (BoT) structure. A BoT can be executed in parallel and in any order in the computation platform [100]. The model aims to optimise the overall BoT execution energy and monetary cost of the hybrid MCC environment under the constraints of mobile device energy and task deadline. A PSO evolutionary algorithm was adopted to find the optimal task scheduling plan [101]. The contributions of this chapter are:

- a model of multi-objective optimisation of device energy and monetary cost in a hybrid MCC environment under the constraints of mobile device energy and task deadline;
- a data-aware offloading technique for data-intensive applications in a hybrid MCC environment; and
- a performance evaluation of the proposed technique using a real experiment and simulations under various working conditions.

The rest of the chapter is structured as follows. Section 3.2 presents an overview of the system architecture as an MCC environment. Application execution models and problem formulations are described in Section 3.3, while the proposed data-aware offloading technique is explained in Section 3.4. The model performance evaluation and experimental results are discussed in Section 3.5. Section 3.6 provides a summary of the chapter.

3.2 System Architecture

This chapter describes a proposed data-intensive mobile application offloading optimisation framework for a hybrid MCC environment. The environment leverages the hybrid MCC resource types.The public cloud offers powerful and scalable resources, while cloudlets are highly accessible resources with efficient computing and storage capabilities which are distributed in geographical locations close to users' devices. These resources can be accessed via WiFi or the cellular network. Mobile devices perform local computation and storage under the constraints of energy availability and wireless interfaces. Section 1.1.2 provides more details about hybrid MCC environment resource types.

The proposed offloading optimisation framework consists of a set of components, described in the following subsections, that provide services context monitoring, decision-making and application execution. Figure 3.1 illustrates the proposed framework.

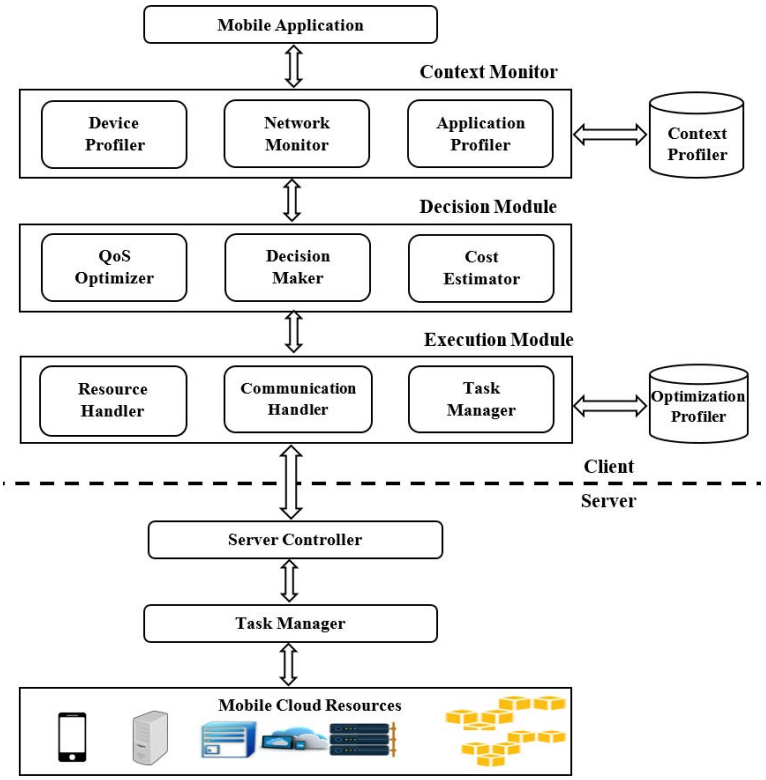


FIGURE 3.1: Proposed hybrid MCC offloading decision optimisation framework.

3.2.1 Context Monitor

The context monitor module is responsible for profiling context parameters at run time and supports the decision-maker with energy consumption and monetary cost estimations. The framework offers three profilers: a device profiler, a network monitor and an application profiler. The device profiler collects data about the user's device, including available energy (battery level), computation power (number of free cores) and free storage space. The network monitor keeps track of the communication between mobile device and remote servers (cloud and cloudlets). Mobile network type and bandwidth are recorded. The data collected about network status is passed to the decision module for data transfer time and cost estimation. The application profiler records profiling data about application execution with awareness of context information about network bandwidth. For each application execution, the profiler registers the offloading decision in association with problem parameters and optimisation values. This allows the decision-maker to estimate

decision variables (time, cost and energy) according to problem parameters like data size and bandwidth. This correspondence is termed data sensitivity analysis.

3.2.2 Decision-Making

The decision-making module includes three components. The cost estimator provides estimations for optimisation variables based on the proposed cost formulation and profiled data by context module.

The QoS optimiser is responsible for evaluating offloading plans according to user QoS constraints of maximum execution time and minimum energy. The evaluation is based on the assumption that each task execution time does not exceed the deadline, and total estimated energy is lower than device available energy. In complex offloading scenarios (with a large number of tasks), the deadline constraint relaxed to allow more tasks to be executed, that is, some tasks can exceed their deadlines by a certain threshold. The threshold refers to the percentage of the original deadline task can exceed to pass the deadline constraint. The selection of the threshold is application-oriented and depends on task time sensitivity.

The cost estimator calculates the values of optimisation decision variables (time, cost and energy) based on the given cost models, and the current device networking status with remote servers. In addition, the cost estimator calculates the expected waiting time for task remote execution based on the adopted queuing system.

The decision-maker runs the offloading optimisation process to find the best application execution plan in a solution space where each solution is estimated by the cost estimator and evaluated by the QoS optimiser. The process can be described as follows:

1. a number of random offloading plans are generated;
2. for each plan, the cost estimator calculates the optimisation cost, and the solution is verified by the QoS optimiser;

3. the resource handler investigates the availability of the required resources;
4. the solution is compared to the current best solution; and
5. the best solution is sent to the execution module.

3.2.3 Execution Module

The execution module is responsible for running and managing the execution of the application execution plan received from the decision-maker. It has three main components. The resource handler manages the communication with computation and storage resources, and is responsible for obtaining details about these resources to determine whether they match the requirements of running application tasks.

The task manager controls the execution of application tasks and manages data sharing through the resource manager. An offloading plan execution is handled as follows:

1. the task manager receives the offloading plan (execution decision) from the decision-maker;
2. the task manager interacts with the communication handler to check the availability of remote servers and then sends remote tasks for execution. The communication handler performs the real data transfer to server controllers on remote servers; and
3. after running all the tasks, plan execution results and execution environment status are sent for profiling.

3.3 System Modelling and Problem Formulation

This section describes the modelling of data-intensive applications and the hybrid MCC environment. Table 3.1 describes the mathematical notations used in the problem formulation.

TABLE 3.1: Problem formulation notations

Symbol	**Definition**
t_i	Application task i
L_i	Task input file location, either locally or remotely
s_i	Task input size
w_i	The number of task execution instructions
∂_i	Task deadline
β_{down}, β_{up}	Available network download and upload bandwidth
β_{cost}	The monetary cost of data communication using the mobile network
d	Mobile device storage
e	Available device energy (joule)
α	Available device memory
w_m	Device processing power
p_{cost}	Cost of processing in a cloud machine
w_{cloud}	Processing power of a cloud machine
C	Estimated total monetary cost of the application
E	Estimated total energy consumption in the mobile device
D_{t_i}	Estimated execution time for task t_i
$\emptyset_i$	Task t_i data size sensitivity factor
β_s, β_r	Network bandwidth between data location and computation target
l	Network latency
$D_{t_i}^W$	Task waiting time in a remote server
λ	Mean rate of arrival of task execution requests at a remote server
L_q	Mean number of task requests in the queue
M^W	Application waiting time

3.3.1 Task Modelling

The representation of a data-intensive hybrid MCC application A is represented as set of independent tasks. An application A is modelled as:

$$A = \{t_1, t_2, \ldots, t_n\} \tag{3.1}$$

where n is the number of tasks. Each task t_i is modelled as:

$$t_i = \{L_i, s_i, w_i, \partial_i\} \tag{3.2}$$

Data size and location parameters are included in task modelling to serve the

objective of building a data-aware optimisation model for scheduling mobile application tasks in a hybrid MCC environment.

3.3.2 Resource Modelling

This section describes the modelling of the hybrid MCC resource model. A mobile device P_m is modelled as:

$$P_m = \{\alpha, \beta_{down}, \beta_{up}, \beta_{cost}, d, e, w_m\} \tag{3.3}$$

A mobile device is connected to a cloudlet and the public cloud via WiFi or cellular networks. A cloudlet or public cloud virtual machine P_{cloud} is modelled as:

$$P_{cloud} = \{\beta_{down}, \beta_{cost}, p_{cost}, w_{cloud}\} \tag{3.4}$$

3.3.3 Application Execution Models

This section describes the cost estimation models involved in formulating the mobile application scheduled to run the application tasks in the hybrid MCC environment. In order to find the application execution plan, three estimation values need to be calculated, namely, task execution time, total mobile energy consumption and total monetary cost.

3.3.3.1 Task Execution Time Model

The task execution time for task t_i is the sum of task processing time $D_{t_i}^P$ in the target computation environment P_{target}, data communication time $D_{t_i}^C$ and task average waiting time D^W for remote execution.

$$D_{t_i} = D_{t_i}^P + D_{t_i}^C + D^W \tag{3.5}$$

$$D_{t_i}^{P} = w_{i,target} + (s_i * \emptyset_i) \tag{3.6}$$

$$D_{t_i}^{C} = \frac{s_i}{min(\beta_s, \beta_r)} + l \tag{3.7}$$

Task remote execution in cloudlets and in the public cloud is modelled as a G/G/1 queuing system [102] with a single computation machine. The queuing system is part of the time estimation process. It requires data from the application profiler and network monitor to estimate the task waiting time based on machine computation availability. For the queuing system, the inter-arrival time of task execution requests has a general distribution. In addition, independent and identical service times among task execution requests with general distribution is assumed. The objective of applying the queuing system is to estimate the average task waiting time D^W. For the G/G/1 queue, no result exists; thus, Shore's approximation is followed [102]. Using Little's rule [103],

$$D^W = \frac{L_q}{\lambda} \tag{3.8}$$

3.3.3.2 Mobile Device Energy Model

The energy consumption of the mobile device E is estimated by calculating E^P, the total processing energy consumed by the mobile device, E^W, the waiting energy (particularly when the local execution time is less than the remote execution time, since the system assumes parallel task execution in the computation environment), and E^C, which is the mobile energy consumption for data transfer communication.

$$E = E^C + E^P + E^W \tag{3.9}$$

$${E_i}^P = {D_{t_i}}^P * {\epsilon_i}^P \tag{3.10}$$

$$E^P = \sum_{i=1}^{m} {E_i}^P \tag{3.11}$$

$$M^W = max(0, \sum_{i=1}^{m} {D_{ti}}^P - max(\sum_{i=1}^{c} {D_{ti}}^P, \sum_{i=1}^{cl} {D_{ti}}^P)) \tag{3.12}$$

$$E^W = M^W * \epsilon^W \tag{3.13}$$

$$E^C = \sum_{i=1}^{m} {D_{ti}}^C * \epsilon^C \tag{3.14}$$

Where:

- $\sum_{i=1}^{m} {D_{ti}}^P$, $\sum_{i=1}^{c} {D_{ti}}^P$, $\sum_{i=1}^{cl} {D_{ti}}^P$: are the total processing times for all tasks executed locally (in the mobile device) and remotely (in the public cloud and cloudlet), respectively. Waiting energy consumption is considered only if the waiting time M^W has non-negative value.
- ${\epsilon_i}^P$, ϵ^W, ϵ^C : the estimated energy consumption in the mobile device for task t_i, remote execution waiting and data communication, respectively.
- m, c, cl are the numbers of tasks to be executed in the mobile device, the public cloud and cloudlet, respectively.

3.3.3.3 Monetary Cost Model

The monetary cost is the amount of money needed to run the mobile application in the proposed hybrid MCC environment. This includes two parts: total remote task processing cost C^P in the cloudlet, and the public cloud and total data communication cost C^C.

$$C = C^P + C^C \tag{3.15}$$

$$C^P = \sum_{i}^{c} {C_i}^P \tag{3.16}$$

$${C_i}^P = {D_{t_i}}^P * p_{cost} \tag{3.17}$$

$$C^C = \sum_{i}^{n} (s_i * \beta_{cost}) \tag{3.18}$$

3.3.4 Problem Formulation

The main objective is to find the best mobile application offloading plan in which the total energy consumption on the mobile device and the total monetary cost are reduced with respect to the individual task deadline and available mobile battery energy. The offloading plan represents a tuple for each task t_i and the selected computation environment for local execution on the mobile device P_m, the cloudlet or the public cloud P_{cloud}. Precisely, the optimisation problem is formulated as monetary cost (C) multiplied by mobile energy consumption (E), due to the assumption of equal contribution to the optimisation and the difference in both measurement units:

$$min(C * E) \tag{3.19}$$

Subject to:

$$D_{t_i} < \partial_i \ , \forall t_i \in A$$

$$E < e$$

3.4 Proposed Data-Aware Offloading Technique

Mobile application offloading aims to augment mobile device capabilities by migrating computation to more powerful resources. Here, a data-aware offloading technique is proposed and the contribution of data size for mobile application offloading decisions in hybrid MCC environment is evaluated. To accomplish this objective, PSO [101] was adopted as an evolutionary search optimisation technique to find the best offloading plan based on the optimisation objective in Eq (3.19). This section discusses the proposed offloading technique, in which an optimised tasks allocation and scheduling plan is generated.

Particle swarm optimisation is an evolutionary computational technique that optimises a problem by iteratively trying to improve a candidature solution with respect to quality and cost. It simulates the behaviour of groups of organisms in motion, such as a flock of birds or school of fish. It was developed by Eberhart and

Kennedy [101] in 1995 and has been widely researched and utilised ever since. A particle represents an individual that tries to find an optimised solution by moving through a defined search space. Each particle adjusts its position towards local best position (own position) and towards global best (the entire population best position). This behaviour improves the convergence time needed to get a global minimum with a reasonably good solution. PSO modelling requires two main steps: problem encoding and fitness function formulation.

Here, a particle represents a randomised application execution plan on available resources. Figure 3.2 provides an example of a particle position. There are 10 tasks to schedule. In this case, a particle is 10-dimensional and its position has 10 coordinates. The coordinate index (coordinate 1 through 10) maps into tasks (t_1 through t_{10}). The value of each coordinate is a real number in the range (0..3]. Coordinate values in the range (0..1] for the corresponding tasks are allocated to the mobile device and coordinate values in the range (1..2] for the corresponding tasks are allocated to the cloudlet, while coordinate values in the range (2..3] for the corresponding tasks are allocated to the public cloud.

To reflect the objective of scheduling tasks in the defined computation environment, the fitness function is used to determine the goodness of a particle's position by estimating the optimisation value for a given solution according to the total monetary cost C Eq. (3.15) and the total energy E Eq. (3.9) consumed by the mobile device.

Algorithm 1 shows the steps for calculating the fitness value for a particle. The fitness function accepts a PSO particle which has a position that represents an application task scheduling solution. The computation process of the offloading

Particle's Position

Crd.1	Crd.2	Crd.3	Crd.4	Crd.5	Crd.6	Crd.7	Crd.8	Crd.9	Crd.10
0.38	1.28	1.86	2.90	2.73	2.63	0.91	2.59	0.47	1.78

Task to Resource Mapping

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
M	CL	CL	C	C	C	M	C	M	CL

FIGURE 3.2: Example of the encoding of a particle's position.

Algorithm 1 PSO fitness function

1: Function PSOFitFun (P, BoT)
2: Inputs: P: a PSO particle, The BoT
3: Output: optimisation value (C*E)
4: $\{C^C, C^P, C, E^C, E^P, E^W, E\} = 0$
5: $\{D^P, D^C, D, D^T M, M^W\} = 0$
6: **for** $i = 1$ to n do **do**
7: $P_target = P.pos(i)$
8: **if** P_{target} != $t_i.L$ **then**
9: D^C = CTFunc(t_i,P_{target})
10: **if** $P_{target} == M$ **or** $t_i.L == M$ **then**
11: $E^C = E^C + CEFunc(t_i, P_{target}, D^C)$
12: $C^C = C^C + CCFunc(P_{target}, D^C)$
13: **end if**
14: **if** $t_i.L == R$ **then**
15: $C^C = C^C + CCFunc(P_{target}, D^C)$
16: **end if**
17: **end if**
18: $D^P = PTFunc(t_i, P_{target})$
19: **if** $P_{target} == M$ **then**
20: $E^P = E^P + PEFunc(P_{target}, D^P)$
21: **else**
22: $D^W = QWFunc(P_{target})$
23: $C^P = PCFunc(P_{target}, D^P + D^W)$
24: **end if**
25: **if** $P_{target} == M$ **then**
26: $D^{TM} = D^{TM} + D\ D^{TM}$
27: **else**
28: $D^{TC} = D^{TC} + D\ D^{TC}$
29: **end if**
30: $D = D^P + D^C + D^W$
31: **if** $D > t_i.\rho$ **then**
32: **return** -1
33: **end if**
34: **end for**
35: $M^W = D^{TM} - D^{TC}$
36: **if** $M^W > 0$ **then**
37: $E^W = WEFunc(P_m, M^W)$
38: **end if**
39: $E = E^P + E^C + E^W$
40: $C = C^P + C^C$
41: **if** $E > P_m.e$ **then**
42: **return** -1
43: **end if**
44: **return** $E * C$

value based on the solution presented by the input practical is summarised as follows.

- The task processing time D^P is calculated based on the computation requirement and data size for task t_i. The impact of data size is measured by performing task processing with different data sizes. The processing time D^P is used to estimate processing energy for local task execution, and monetary cost C^P for remote execution.

- Energy estimation and cost calculation are based on task execution time D Eq. (3.5), including processing, communication and waiting time.

- The system assumes extra energy consumption as the mobile device is in idle state. This is referred to energy as device waiting energy, and is calculated when total local execution time is less than the maximum remote execution time in the cloudlet or public cloud, line 37.

- The data communication time D^C for task t_i depends on task data location and is considered only if the data storage and computation environment are in different locations, lines 10-16.

- The fitness function considers the impact of D^C time on mobile device energy and monetary cost.

- In the case of remote execution, the waiting D^W is estimated using the queuing system and calling the function $QWFun$ in line 22.

- Model constraints for deadline and available energy are checked in lines 31 and 36 respectively.

Algorithm 2 shows the main steps of finding the optimal offloading scheduling plan for data-aware mobile applications. Initially, the system collects data about the computation resources R (mobile device and cloud resources). For the mobile device, the available energy and available bandwidth are collected. The first step is PSO environment configuration setup based on standard values of the parameters of inertia and coefficient acceleration. Next, PSO particles are initialised with random position. A particle position represents a BoT execution plan on computation resources.

Algorithm 2 Task scheduling for data-aware offloading

```
1: Inputs : set of application tasks T, computation resources R
2: Output : application tasks schedule S
3: Update resources' metadata
4: Setting up PSO Environment P
5: Initialise PSO Particles P[NumP] NumP : number of particles
6: P.gbest = inf Initialise global best gbest
7: for i = 1 to NumP do
8:    Randomise P[i].POS
9:    FitCost = PSOFitFun(P[i].POS, T)
10:   UpdateBestPos (P, P[i], FitCost)
11: end for
12: Run PSO Iterations // NumL : number of iterations
13: for i = 1 to NumL do
14:   for j = 1 to NumP do
15:      Calculate P[j].VELOCITY
16:      Update P[j].POS
17:      FitCost = PSOFitFun(P[j].POS, T)
18:      UpdateBestPos (P, P[j], FitCost)
19:   end for
20: end for
21: MinCost = PSOFitFun(P.gbest.POS, T)
22: S = (P.gbest.POS, MinCost)
23: Return S
```

For each particle, the fitness cost is calculated based on the fitness function provided in Algorithm 1. The local best *pbest* and global best *gbest* are updated. The *gbest* is initialised to infinity for this cost minimisation problem. To find an optimal solution, the algorithm performs multiple iterations, and in each iteration, each particle position is updated based on its new velocity value. *pbest* and *gbest* values are updated to reflect the new fitness cost estimation. Once the algorithm finishes all iterations, the particle associated with *gbest*, which represents the minimum optimisation value of $E*C$, according to the optimisation objective, will be considered as the optimal BoT execution plan.

3.5 Performance Evaluation

Performance evaluation for the proposed data-aware offloading technique was conducted using two experiments. The first was a real experiment, designed to validate

the model by comparing the results of the offloading technique against real execution. This experiment required a real MCC environment, as illustrated in Figure 3.1. In the second experiment, synthetic application data was used to evaluate the proposed offloading technique.

3.5.1 Experimental Setup

The configuration of the hybrid MCC resources includes mobile devices, cloudlets and the public cloud listed in Table 3.2. It assumes a single machine for task processing in the public cloud and cloudlet. For mobile network bandwidth, the experiment was conducted with three communication networks (3G, 4G, and Wi-Fi). Table 3.3 presents details about bandwidth values for these networks. Minimum and maximum bandwidth in real application execution were recorded for each network type and used to build a uniform distribution model for the experimental work. The application structure was initialised with 30 tasks. Each task has the following properties.

- The computation requirement (task workload) D^P is based on the workload model proposed by Anglano and Canonico [104]. Application task deadline values are uniformly distributed in the interval $[X \pm 0.5X]$, where X is the granularity of the tasks. The model was used to determine the task deadline

Table 3.2: Computation resources configuration

Resource type	**No. Cores**	**Memory (GB)**
EC2 Linux t2.2xlarge Intel Xeon	8	32
Cloudlet Intel Xeon	4	8
LG Nexus 5 Qualcomm	2	2

Table 3.3: Communication Networks bandwidth

Network Type	**Min. Bandwidth (MB/s)**	**Max. Bandwidth (MB/s)**
3G	1.5	2.6
4G	3.9	9.5
WiFi	10.8	20.5
Network Latency	**Min. Latency (s)**	**Max. Latency (s)**
Latency	0.6	11.5

with $X = 1000$ granularity. After applying the model, deadline values are uniformly distributed between 500 and 1500, all in seconds.

- The task data file locations (L) are distributed randomly between the public cloud server, the mobile device and other cloud storage (i.e. AWS S3).

- Task data size (s) model: four different scenarios for the application data model are employed to create each task. Table 3.4 presents four task data distributions with minimum and maximum sizes. The data distributions are selected to align with the study of data size contribution on offloading optimisation decision.

- The task data sensitivity factor (ω_i) has a value between [0,1] and measures the task execution response to the change in data size.

TABLE 3.4: Task input data size distribution

Data Distribution	**Min. Size (MB)**	**Max. Size (MB)**
Small	20	200
Medium	200	500
Large	500	2000

3.5.2 System Profiling

This section outlines the implementation of a profiling process to compute an approximation of energy and bandwidth estimation parameters in a hybrid MCC environment. The profiling process focused on estimating the relationship between model parameters of network bandwidth and data size and the energy consumed by the user's mobile device. A profiling experiment with varying data size (different input data files) and bandwidth (change mobile network) was performed. At each execution setup, a 30-tasks BoT application was run. Averages of processing, communication and waiting energy values were recorded. PowerTutor software was used for power estimation, giving energy consumption estimates within 5% of actual values [105].

Table 3.3 shows bandwidth profiling results. In addition, energy parameters were used to calculate the energy the device consumed during task processing, data transfer and device waiting for external task processing, ${\epsilon_i}^P$, ϵ^C, ϵ^W, respectively. The profiling process considers the parameters of task data size, mobile network, device processing capability and physical data file transfer between data storage and computation locations. The experiment was conducted for each combination of data size and bandwidth variations. Table 3.4 shows the data size distributions for data files used in the profiling experiments.

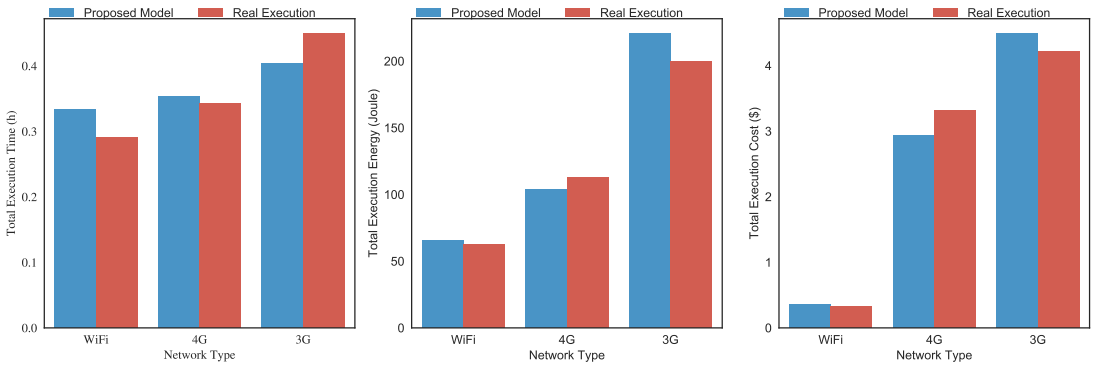
In addition, as a part of system profiling, the data sensitivity factor ω_i for a task t_i was estimated. The sensitivity factor measures the sensitivity of task processing to change in task data size. To get an accurate measurement of the sensitivity factor, the correlation between task input data size and the change in processing time was analysed. Thirty experiments were performed to reach a reliable measurement with minimum variance on data sensitivity values.

The following provides the validation for the proposed execution model using a real-time execution scenario. Validation was based on running an application with 30-task of small data model using both the proposed model and real implementation for three communication networks.

Figure 3.3 shows the proposed model and real execution results in three communication networks and three performance metrics: execution time, mobile energy and monetary cost. For the purpose of validation, only the small data scenario was implemented. The validation process can be described as follows.

1. Set up the environment. The execution environment includes three computation resources: a cloud server, a cloudlet and a mobile device. We have applied the same experiment setup for profiling environment configuration. This includes resources setup which is provided in Table 3.2 and communication network setup which is provided in Table 3.3.

2. Construct 30 execution scenarios for the 30-task BoT.



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 3.3: Application execution model validation using real experiments for three different communication networks

3. For each network scenario, run the offloading optimisation to find the execution plan for each execution scansion.

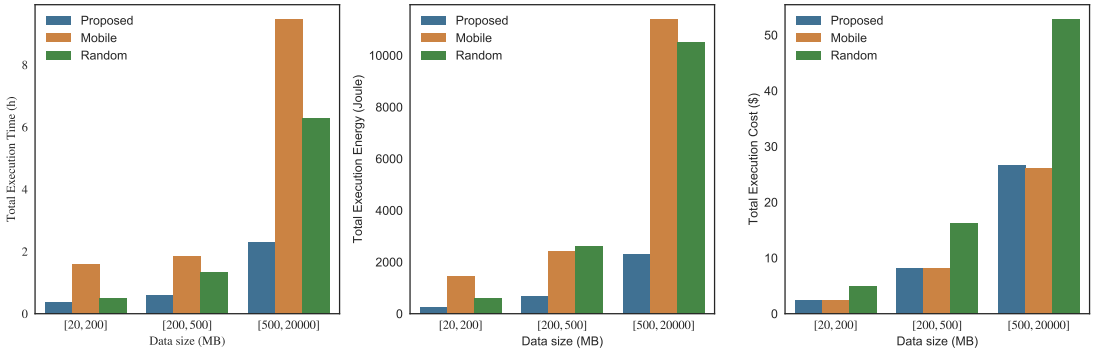
4. Run the offloading plan on the real execution environment and record results.

The results shows some discrepancies between the proposed model and real execution scenario due to the differences in available bandwidth and network latency. For execution time in Figure 3.3, the proposed model scenario demonstrates higher values than the real-time with high-bandwidth networks. This can be explained by the variation in bandwidth distribution interval which feeds the offloading system with bandwidth values. This behaviour does not apply for small bandwidth interval such as in 3G case. Another source of execution time overestimation is the applied queuing system to estimate task waiting time at remote servers. For the purpose of experimentation, a uniform distribution is adopted for each server service time. Overall, the highest estimation error was observed in the 3G network. As shown in Table 3.3, the bandwidth for the 3G network is between 1.5 and 2.6, but in real execution the bandwidth is unstable. The average errors for the execution model are 8% for execution time, 11% for energy consumption and 15% for monetary cost. Based on average estimation errors, it can be concluded that the proposed execution model is accurate enough to use for offloading in the hybrid MCC system.

3.5.3 System Evaluation and Experimental Results

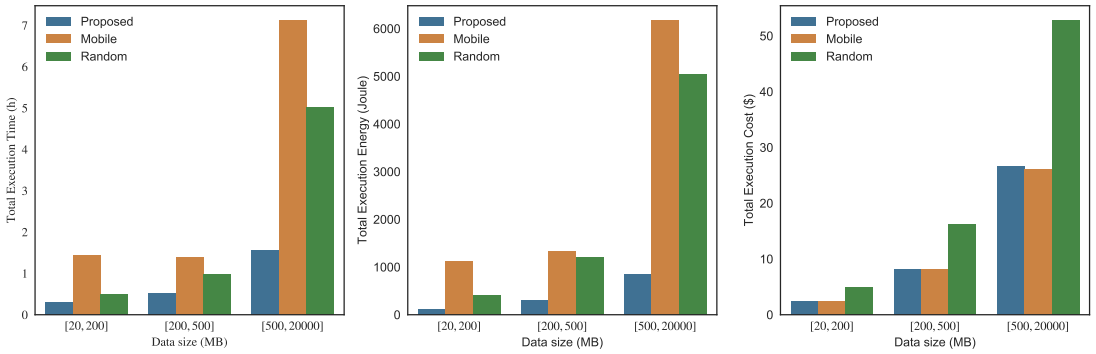
After validating the proposed offloading model, a similar setup was used to evaluate the proposed offloading technique under varying working conditions. The proposed technique was compared with two other techniques: mobile only and random offloading. The former implies only local processing on the user's device, and the latter refers to the selection of random offloading using a combination of mobile, cloudlet and public cloud. The application tasks were demonstrated based on the aforementioned data size scenarios. In addition, the performance measurements used were execution time, mobile energy and monetary cost. Figures 3.4, 3.5 and 3.6 show the results of running the mobile application using three techniques and three communication networks, giving nine sets of results in total.

Figure 3.4 shows the experimental results when using the 3G network. Figure 3.4 (a) illustrates that the proposed technique reduces the execution time in comparison to the other two techniques for the three data size models. In comparison to the random technique, execution time reduction was 25% with small data size, 56% with medium data size and 63% with large data size. For the same experiment, the proposed technique, compared to the mobile technique, reduced the execution time by an average of 73% for the three data size models. Moreover, Figure 3.4 (b) shows reduction in mobile energy consumption with increasing data size. For example, the proposed technique reduced mobile energy consumption by



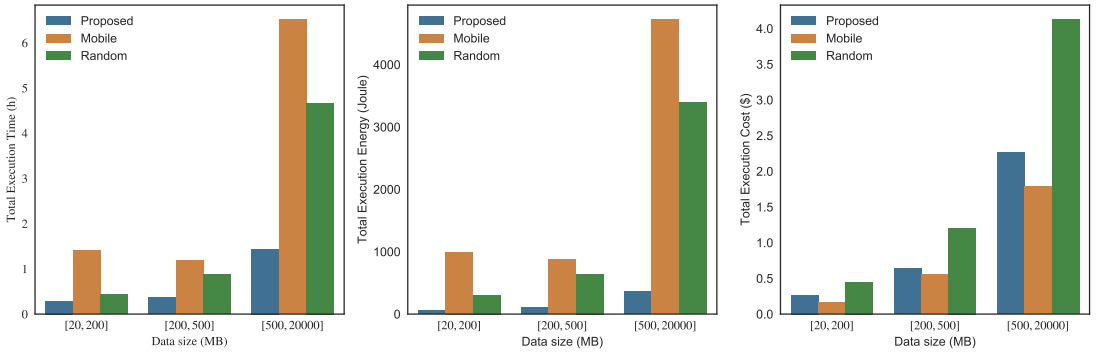
(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 3.4: System performance measurements with 3G network for different data sizes



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 3.5: System performance measurements with 4G network for different data sizes



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 3.6: System performance measurements with WiFi network for different data sizes

57% with small data size, 74% with medium data size and 78% with large data size compared with the random technique.

The proposed technique, compared to the mobile technique, reduced the mobile energy consumption by an average of 78% for the three data size models. In addition, Figure 3.4 (c) highlights the ability of the proposed technique to reduce monetary cost by 2-3% compared to the mobile technique, and 50% compared to the random technique. This is due to the proposed technique's execution in the mobile environment incurring only communication costs' while the other two techniques have communication costs and cloud resources costs. In general, as data size increases, the ability of the proposed technique to handle data-intensive applications also increases.

In the 4G case Figure 3.5, results confirm the proposed technique behaves similarly to the random and mobile techniques. Figure 3.5 (b) shows energy consumption savings of the proposed technique compared with the mobile technique of 89% for large data size; this compares to 80% for the 3G network, due to the higher energy consumption of the 4G network. An important conclusion in regard to the use of cellular networks is the proposed technique's efficiency in providing cost-effective offloading plans by using the capability of nearby cloudlets and reducing the amount of computation and data migrated to the cloud.

The impact of low-cost networks on computation time and cost is represented clearly in the case of WiFi Figure 3.6. The proposed technique's strategy is to maximise dependency on cloud resources. Thus, Figure 3.6 (a) shows an increase in total execution time, and (b) shows average 90% and 84% savings in mobile energy consumption of the proposed technique compared with the mobile technique and the random technique, respectively. This again confirms the significance of the proposed model in saving energy while using different communication networks.

Based on these results, the proposed technique improves execution time for data-intensive applications for large data sets compared to the random technique by an average of 51% and reduces mobile energy consumption by an average of 77%, while the average monetary saving is 48%. Moreover, the proposed technique improves execution time for data-intensive applications for large data sets compared to the mobile technique by an average of 78% and mobile energy consumption by an average of 87%, while the average monetary cost is only 11% higher due to using cloud resources and mobile communication.

In summary, the proposed technique shows greater ability to handle data-intensive applications than the mobile and the random techniques through reducing mobile energy consumption and monetary cost. It can be stated that for data-intensive applications, using the proposed technique is superior because the execution time and mobile energy consumption will be very high, but for small data size applications, it has minimal advantages over the mobile technique and the random technique.

3.6 Summary

This chapter outlines a proposed QoS-aware resource allocation model for scheduling data-intensive mobile applications in a hybrid MCC environment. The model generates an application execution plan that consider application complexity, input data size, available network bandwidth, and available mobile device energy. Results indicate that the model reduces execution time and energy consumption substantially. However, the execution model accuracy for estimating waiting time on remote servers can be improved by more investigation on the queuing system behaviour and incorporating additional parameters like task arrival rate and task service rate.

One issue with evolutionary optimisation techniques like PSO is the exponential increase in search complexity, particularly with large-space problems such as mobile applications with large numbers of tasks. The next chapter describes an investigation of offloading optimisation with linear search optimisation to reduce search time complexity. In addition, it outlines a study of the integration of edge computing into the hybrid MCC environment to improve handling of data-intensive MCC applications.

Chapter 4

Data-Intensive Application Scheduling on Mobile Edge Cloud Computing

Chapter 3 describes an investigation of the role of hybrid MCC in overcoming the challenges of mobile computing by allowing mobile devices to offload computation-intensive and data-intensive tasks to high-performance and scalable computation resources. However, emerging data-intensive applications pose challenges for hybrid MCC platforms because of high latency, cost and data location issues. To address the challenges of data-intensive applications on mobile cloud platforms, this chapter describes a proposed application offloading optimisation model that schedules application tasks in an MECC environment. The optimisation model is formulated as a MILP model, which considers both monetary cost and device energy as optimisation objectives. Moreover, the allocation process considers parameters related to data size and location, data communication costs, context information and network status.

To evaluate the performance of the proposed offloading algorithm, we conducted real experiments on the implemented system with a variety of scenarios, such as different deadline and multi-user parameters. The results demonstrate the ability

of the proposed algorithm to generate an optimised resource allocation plan in response to dramatic fluctuations in application data size and network bandwidth. The proposed technique reduced the execution cost of data-intensive applications by an average of 46% and 76% in comparison with PSO and full execution on a mobile device only, respectively. In addition, the new technique reduced mobile energy consumption by 35% and 84%, compared to PSO and full execution on a mobile device only, respectively.

4.1 Introduction

MCC aims to augment the capacity of mobile devices by improving and optimising their computing capabilities by performing computationally intensive tasks using cloud-based resources [25], and involves migrating resource-intensive computations from smartphone devices to the cloud via wireless communication technologies, a process referred to as the computation offloading concept.

In the previous chapter, a data-intensive offloading technique on hybrid-MCC based on the PSO optimisation technique was proposed. The technique has two main limitations. The first limitation is handling the challenges of delay-sensitive applications, where the long propagation distance from a mobile device to a remote cloud server can result in an excessively long latency for mobile applications, and the other corresponds to the high time complexity of optimising large-scale problems using PSO technique. As discussed in Section 1.2, alternative computing models like MEC can handle the latency issue by allowing approximate computation closer to data sources [40].

Mobile edge computing is an emerging paradigm designed to meet the ever-increasing computation demands of mobile applications [66]. Edge computing is a computing paradigm designed to facilitate the management of heterogeneous data sources for collection, processing and transfer of massive data generated from interconnected devices [106]. Applying edge computing can be sufficient to reduce the overload of data flow requested to the cloud. Moreover, offloading some computing tasks

to the edge can improve energy efficiency due to the huge data transfer saving, particularly with mobile network usage [30]. Most studies of offloading optimisation in MEC focus on finding an optimal application task distribution strategy to edge resources that minimises data transfer latency with respect to the capacity of edge nodes [29, 34, 40, 67–70].

Mobile edge computing has the advantage for data-intensive to push mobile computing, network control and storage from resource-limited mobile devices to the network edges to enable computation-intensive and latency-critical applications [35]. With MEC, a mobile application can track real-time information such as behaviours, location and environment, which reduces the exchange of sensitive information between the mobile device and cloud resources, while being more energy efficient. However, edge resources are limited in the computation capacity, scalability and high-energy sensitivity needed to perform long-term computation. One solution is integrating edge and cloud resources, providing benefits from the high capability of cloud resources and the availability of access resources at the edge layer. The joint computation architecture can be referred to as MECC [41].

In this chapter, multi-objective data-intensive mobile application scheduling and allocation optimisation in an MECC computation environment is proposed. The contributions of this work include:

- multi-objective optimisation of device energy and monetary cost in the MECC environment under the constraints of available mobile device energy and task deadline;
- formulation of joint task offloading and resource allocation optimisation for multi-user mobile applications using MILP, considering both monetary cost and device energy as optimisation objectives; and
- a multi-perspective analysis of data-intensive application optimisation based on deadline and multi-user constraints.

This chapter is structured as follows. Section 4.2 describes an investigation of related work on task scheduling and allocation optimisation on MCC, MEC and

MECC. Section 4.3 presents an overview of the system architecture. System modelling and problem formulation are presented in Section 4.4, while the optimisation technique and the proposed offloading algorithm are explained in Section 4.5. The proposed model's performance evaluation and experimental results are discussed in Section 4.6, and Section 4.7 provides summary of the chapter.

4.2 Related Work

This section contains a review of the existing work on task scheduling and allocation optimisation using MCC, MEC and MECC from different perspectives and based on different optimisation objectives, including, energy, cost, latency and multi-user implementation. Mobile device I/O processing and network communications are energy-hungry components [21], and offloading heavy tasks to high-performance computation resources can substantially reduce energy consumption. Many techniques have been proposed for overcoming the challenge of energy-intensive applications, such as device augmentation [21], VM cloning [59], computation migration [74] and code decomposition and component reusability [22]. Despite their capacity to fulfil the requirements of some applications, like peer-to-peer gaming and low-scale image processing, these techniques encounter major challenges in supporting large-scale and complex applications with unpredictable numbers of users, exchanged data size, network bandwidth and corresponding data communication costs.

Mobile/wireless network performance makes a significant contribution to mobile application responsiveness and processing time [15]. Hung et al. [107] argued that a high percentage of mobile users would prefer to run mobile applications locally due to network performance implications. The decision to run an application locally or remotely is complicated and requires steady monitoring of network conditions and application profiling [61]. Many researchers have focused on resolving the issues of instability of network bandwidth and data communication quality.

However, one limitation of cloudlets is their limited convergence on the mobile network for service provisioning, which does not support a high number of mobile users sharing available resources [81]. Processing on close edge nodes is more efficient for reducing data transfer latency and capturing more real-time context information [34]. Chen et al. [81] adopted MEC architecture and applied game theory to find optimised application scheduling based on latency and energy constraints. Jararweh et al. [68] addressed the same optimisation problem using MILP in an edge-only resource environment. Cardellini et al. [85] designed a game-theoretic approach for computation offloading in a joint resource model of cloud and edge computing. The work employed the waiting time in computation nodes to separate the execution of multi-user applications without consideration of user workload distribution. Offloading decisions on cloudlet-based architecture needs to consider the level of interactivity and workload dynamicity among system users.

In the context of data-intensive mobile applications, some issues related to MECC are not clearly resolved. Firstly, transferring huge amounts of data via unstable mobile networks will lead to an unpredictable increase in data transfer latency. Secondly, edge resources have limited capability to process application tasks with high data input. Finally, transferring massive data over cellular networks and processing in public clouds incurs significant costs for application users. Wang et al. [18] argued that data-intensive application computations on mobile computing are always costly in terms of computation time, mobile energy and resource cost. Nan et al. [51] studied the challenges of data-intensive-aware mobile applications highlight the significance of increasing data size on monetary cost and QoS improvement. Abbas et al. [29] proposed an offloading optimisation technique-based execution path to reduce execution latency for data-intensive mobile applications in MEC, and Terefe et al. [80] proposed a data-intensive energy-efficient optimisation model for a hybrid-cloud environment. The results of both studies show how adopting a heterogeneous resource model can minimise energy consumption. Zhou et al. [90] proposed a three-tier MCC middleware, mCloud, that empowers programmers with computation alternatives, based on the application cost model

and the offloading decision-maker. Even though the proposed model includes the task data size in generating an optimised execution application task plan, the data size is relatively small and cannot reflect the scenario of data-intensive application task scheduling on MCC. Moreover, the mCloud offloading optimiser relies on reducing the execution time and energy consumption. Our optimisation model incorporates the monetary cost as an essential quality measurement when using commercials clouds and mobile data networks.

The literature includes efforts to overcome the challenges of data-aware mobile application offloading; however, the proposed models do not consider the contribution of data size variation in association with other optimisation parameters in application offloading decisions. In addition, these models largely neglect the integration of cloud and edge computing to resolve data-intensive application offloading optimisation and fail to address certain other issues. These include the efficient adoption of MEC to overcome latency-sensitive application when transferring and processing large data files and offloading optimisation in response to new scenarios, such as deadline and multi-user models. In this chapter, an optimisation model for offloading data-intensive mobile applications in MECC is proposed. The model adopts a MILP technique to construct an allocation strategy, reducing energy consumption and total monetary cost for the mobile user under the constraints of task-level deadline and available device energy.

4.3 System Architecture

In this section, a description of the MECC environment is provided, followed by the main optimisation engine components. Figure 4.1 illustrates the proposed MECC resource model and optimisation framework. The MECC environment leverages three resource layers, namely, public cloud, edge and mobile devices. The public cloud is a powerful and scalable resource that allows for convenient processing and storage capabilities for computation-intensive and data-intensive tasks. At the edge layer, a user can access nearby computation and storage units with low

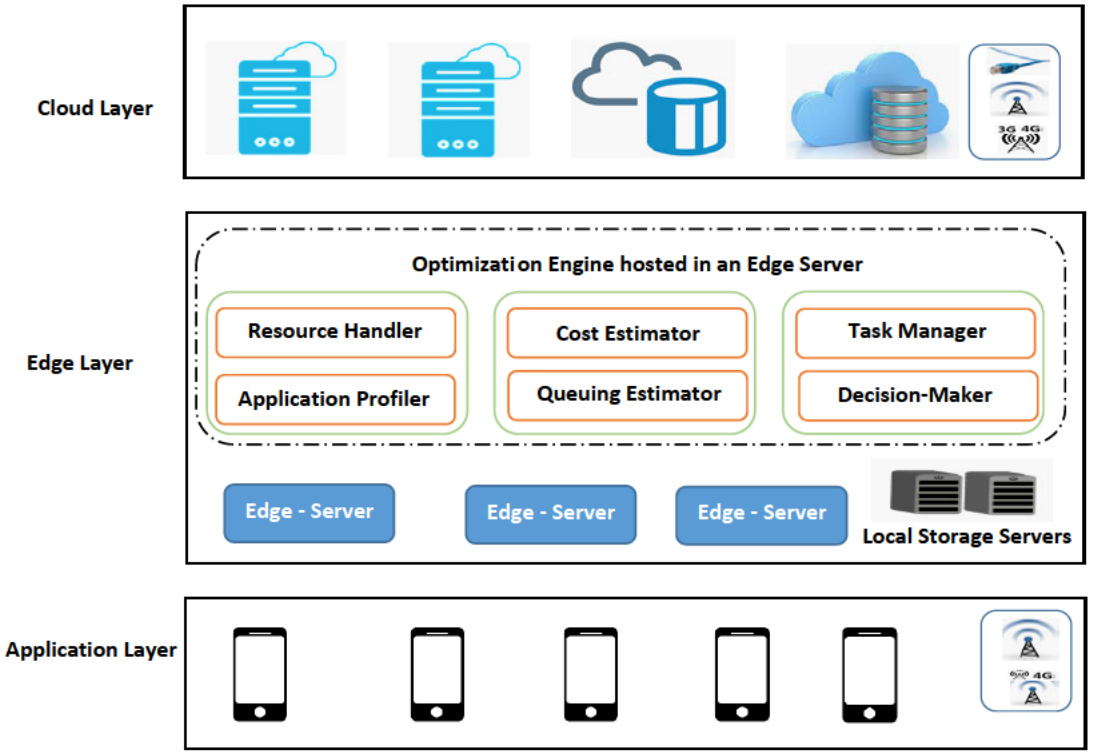


FIGURE 4.1: Mobile-edge cloud computing (MECC) framework

latency time via mobile cellular networks (3G, 4G or WiFi). This research proceeded under three assumptions. First, a low to medium computation power of edge machines in comparison to those in the public cloud. Second, at the edge layer, the availability of local data storage units that are capable of storing large application files. Third, at the application level, the ability to execute tasks locally on users' mobile devices under the constraint of available device energy. The optimisation engine is responsible for handling all activities to construct the offloading/allocation plan for application tasks including profiling, cost and queuing estimation, and decision-making. The optimisation process itself is not resource intensive. Both cloud and edge servers can be used to eliminate the energy overhead to perform the optimisation process on the user device. However, the edge is selected because it has lower latency for decision-making than the cloud. The optimisation engine consists of six components categorised into three service levels: context data collection (application profiler and resource handler), optimisation variables estimation (cost and queue estimator) and offloading decision optimisation (QoS optimiser and decision-maker). These components are described as

follows.

4.3.1 Application Profiler

The framework offers three types of profiling, namely, energy profiling for energy usage, network profiling to monitor mobile network information, and application execution profiling to record profiling data about application execution with awareness of contextual information about network and bandwidth. The application profiler keeps a record of application execution at task level. profiling data include type of application, context information, including device energy, network interface and bandwidth; and optimisation result, including execution time, cost, energy, and task allocation plan. The profiling data is stored in an accessible database to be shared with estimation components (cost and queuing estimators).

4.3.2 Resource Handler

The resource handler works on collecting data about the current status of user device energy and edge resources. Collected data includes device energy level, in-use mobile network and current network bandwidth, number of available cores on mobile and edge nodes, and the connectivity status of the mobile device and edge nodes. The resource handler fetches resource status prior to the optimisation process to be recorded by the application profiler. In addition, the resource handler is responsible for communicating with computation nodes and storage units to run the application according to the optimisation plan. In other words, the resource handler performs all communication aspects with external entities for computation, storage and profiling.

4.3.3 Queuing Estimator

A limited number of cloud servers and edge nodes is assumed. Thus, a queuing system is adopted to support the application offloading decision and manage task

offloading to these resources with consideration of server capability and task waiting time. The work represents computation servers at cloud and edge layers as G/G/1 queues [108]. The queue estimator computes the task waiting time in each processing queue, based on the current distribution of application tasks and the capability of the computation server (service time).

4.3.4 Cost Estimator

The cost estimator handles the calculation of optimisation parameters, that is, execution time, monetary cost and expected energy consumption. The cost estimator communicates with the application profiler to get updated energy profiling parameters for task processing, data transfer and device waiting for external task processing.

4.3.5 Task Manager

The task manager coordinates the execution of application tasks based on the offloading plan generated by the decision-maker. It sends task executions through the resource handler, and updates the application profiler with execution results.

4.3.6 Decision-Maker

The decision-maker is responsible for generating execution plans (paths) and then selecting the optimised one, based on the optimisation constraints of available device energy and task deadline. The decision-maker runs the optimisation process based on resource data from the resource handler and estimation values from cost and queuing estimators.

To avoid the complexity of distributed file storage, it is assumed that each task is associated with a single file storage and the file storage is consistently available for data collection and is capable of storing all task data files. The available storage

in the edge layer is temporary and just for input data and temporary data needed by tasks. This assumption about data storage at the edge layer is illustrated in the proposed system architecture, Figure 4.1. Data files are stored in temporary centralised local storage servers at the edge layer to handle the complexity of large data file access by computation resources, which allows the execution of data-intensive applications such as data analytics, online monitoring and social sensing. In addition, there is long-term storage in the cloud layer, which is used to store required data for all applications. However, the assumption of consistent task data availability cannot be applied at all data collection scenarios, particularly when data is collected at real time manner. Thus, more sophisticated file management and data replication schemes need to be injected. The research and development towards these schemes is a promising future research direction.

4.4 Application Model

This section describes the modelling of mobile application execution in an MECC system as a three-tiered computing system of mobile devices (application user), edge resources and cloud server. Table 4.1 describes the mathematical notations used in the problem formulation.

Table 4.1: Problem modelling notation

Symbol	**Definition**
t_i	Application task i
L_i	Task input file location, either local or remote
s_i	Task input size
I_i	The number of task execution instructions (MIPS)
∂_i	Task deadline
β	Available network bandwidth
β_{cost}	The monetary cost of data communication using a mobile network interface
w_i	Processing power for mobile devices or remote computation nodes
d_m	Mobile device storage (MB)
e_m	Mobile device available energy (J)
p	Cost of processing in a remote execution server ($/hour)
ω_i	Task t_i data size sensitivity factor
l	The network latency

4.4.1 Task Model

A data-intensive mobile application A is represented as a BoT, in which tasks are independent in computation and data sharing. The BoT application model allows a variety of data locations (to test the bandwidth impact on data transfer) and a variety of data sizes. In BoT, each task can have its own computation complexity, which allows better linkage between data-intensive and computation-intensive application features. BoT is a common application model for large-scale systems, such as computational biology, parameter sweeps, fractal calculations and data mining [109]. An application A is modelled as:

$$A = \{t_1, t_2, \ldots, t_n\} \tag{4.1}$$

where n is number of tasks. A task t_i is modelled as:

$$t_i = \{L_i, s_i, I_i, \partial_i\} \tag{4.2}$$

4.4.2 System Model

The computing system assumes three computation environments: public cloud, M edge nodes and N mobile devices. A mobile device P_m is modelled as: $P_m = \{\beta, \beta_{cost}, d_m, e_m, w_m\}$. A mobile device is connected to an edge and the public cloud via WiFi or cellular networks. A remote computation node on edge or public cloud P_r is modelled as: $P_r = \{\beta, \beta_{cost}, p, w_r\}$. Assume that, in a given time slot, a mobile user can request application execution; some tasks will be executed locally (on the mobile user device) and others remotely (on an edge node or the cloud server). A joint offloading technique is adopted to schedule application tasks in the MECC computation environment. In addition, in this work, cloud and edge resources are assumed to be available and pre-allocated (static provisioning) by the platform (dynamic provisioning is an interesting aspect for future research). Thus, rather than adding cloud resources to process extra tasks arriving in the

system, the process adopts a queuing system in which tasks wait for resources. This model minimises the monetary cost of cloud resources.

The next section describes the cost estimation models involved in formulating a response to the mobile application scheduling and allocation problem. To find the application execution plan, three value estimations need to be calculated, namely, task execution time, total energy and total monetary cost.

4.4.3 Task Execution Time Model

The task execution time for task t_i is the sum of task processing time D_i^P in the target computation environment P_r or P_m, data communication time D_i^C and task average waiting time D_i^W for remote execution. However, the task processing time D_i^P depends on the number of task instructions I_i to run the task and the target computation unit power w_{target}. Moreover, the task data size is considered in calculating the task processing time. To do so, a data sensitivity factor ω_i, which is calculated as the ratio of change in processing time to the change in data size, is introduced. To get an accurate measurement of the sensitivity factor, the correlation between the task input data size and the change in processing time was calculated. Several experiments were conducted to reach a stable measurement with minimum variance on data sensitivity values.

$$D_i = D_i^P + D_i^C + D_i^W \tag{4.3}$$

$$D_i^P = \frac{I_i}{w_{target}} + (s_i.\omega_i) \tag{4.4}$$

$$D_i^C = \frac{s_i}{\beta} + l \tag{4.5}$$

To compute the task waiting time for task t_i, a G/G/1 queuing model is adopted. The G/G/1 queue represents the queue length in a system with a single server where inter-arrival times have a general (arbitrary) distribution and service times have a (different) general distribution. Edge and public cloud computation resources are assumed to be G/G/1 queues, in which the task arrival rate λ for

processing follows a general distribution and the service time μ to process incoming tasks similarly follows a general distribution. Marchal [108] proposed an approximation for G/G/1 as follows:

$$L_q \approx \frac{\rho^2(1+C_s^2)(C_a^2+\rho^2 C_s^2)}{2(1-\rho)(1+\rho^2 C_s^2)} \tag{4.6}$$

$$C_s^2 = \frac{\sigma_s^2}{(1/\mu)^2} \tag{4.7}$$

$$C_a^2 = \frac{\sigma_a^2}{(1/\lambda)^2} \tag{4.8}$$

where L_q is the queue length, ρ is the server utilisation, $\rho = \frac{\lambda}{\mu s}$, $s = 1$ is the number of servers, C_s^2 is the coefficient square of the service time variance, C_a^2 is the coefficient square of the inter-arrival time variance, σ_s^2 is the variance of the service time and σ_a^2 is the variance of inter-arrival time. Using Little's rule, the queue waiting time (D_i^W) can be computed as:

$$D_i^W = \frac{L_q}{\lambda} \tag{4.9}$$

4.4.4 Mobile Device Energy Model

The energy consumed by mobile device E_i to execute t_i is estimated by calculating the total processing energy E_i^P consumed by the mobile device, the waiting energy E_i^W and data transfer energy E_i^C.

$$E_i = E_i^P + E_i^C + E_i^W \tag{4.10}$$

$$E_i{}^P = D_i{}^P.\epsilon_i{}^P \tag{4.11}$$

$$E_i^C = D_i{}^C.\epsilon^C \tag{4.12}$$

$$E_i^W = D_i{}^W.\epsilon^W \tag{4.13}$$

Where $\epsilon_i{}^P$, ϵ^W, ϵ^C are the estimated energy consumption per second in the mobile device for task t_i, remote execution waiting (in seconds) and data communication

MB/S, respectively.

4.4.5 Monetary Cost Model

The monetary cost represents the amount of money required to run a task t_i in a target computation environment P_r. This includes two parts. The first part is the task processing cost C_i^P in a remote server at edge nodes or the public cloud. The processing cost is measured using the processing time length D_i and the cost per hour for the host server p_i. The second part is the data communication cost C_i^C, which is measured by the amount of data s_i to be transferred with bandwidth cost β_{cost}.

$$C_i = C_i^P + C_i^C \tag{4.14}$$

$$C_i{}^P = D_i{}^P.p_i \tag{4.15}$$

$$C_i^C = s_i.\beta_{cost} \tag{4.16}$$

4.5 The Proposed Offloading Algorithm

The system objective is to find the optimal solution in which the total energy consumption of the mobile device and the total monetary cost are minimised under the constraints of available device energy and task execution deadline. The optimisation algorithm is implemented in the decision-maker, which is responsible for collecting the information needed to run the algorithm and produce the optimisation solution. The solution represents a tuple of each task t_i and the selected computation environment, either local execution on the mobile client device P_m or remote execution on external computation machine P_r (edge nodes or public cloud VM).

This work utilises the MILP technique to address cost, energy and time optimisation on MECC architecture. A MILP formulation essentially determines that all objective functions and constraint equations are linear. The linear nature of

MILP facilitates the easy and rapid solution of any subproblems, while allowing for relatively complex formulations. Such an approach is usually faster and less computation intensive than a non-linear technique. Additionally, MILP's linear nature ensures that any minimum obtained is a global minimum and not a local one [110]. Furthermore, the complexity time for MILP increases linearly as the problem space grows. On the other hand, the MILP algorithm has some limitations with respect to the nonlinearity effects and the high dimensionality of the given problem [111]. For the system and workload considered here, the proposed MILP technique is able to find the optimal solution. However, for more complex workload (e.g., applications with complex dependencies or variable data size), MILP might be able to generate only suboptimal solutions. This could be part of future work on non-linear methods involving quantitative comparisons.

The binary offloading decision variable for task t_i is denoted by

$$x_i = (x_i^l, x_{i1}^f, ..., x_{iM}^f, x_i^c) \tag{4.17}$$

In which x_i^l, x_{ij}^f, x_i^c indicate that task t_i is processed locally at the mobile device, edge node ($1 \leq j \leq M$) or the cloud server, respectively. A solution x_i represents the binary encoding for task t_i allocation on system resources. Only one position is set to number 1, which indicates that the corresponding machine is allocated that task. Remaining positions are set to number 0. Based on solution x_i for task t_i, optimisation parameters, that is, execution time D_i, energy consumption E_i and monetary cost C_i are calculated as follows.

$$D_i = h_i^T x_i, E_i = e_i^T x_i, C_i = c_i^T x_i \tag{4.18}$$

where

- $h_i = (T_i^l, T_{i1}^f, ..., T_{iM}^f, T_i^c)$ (the calculated end-to-end execution time for task t_i on each computation unit). See Eq. (4.3);
- $e_i = (E_i^l, E_{i1}^f, ..., E_{iM}^f, E_i^c)$ (the calculated end-to-end energy for task t_i on each computation unit). See Eq. (4.10); and

- $c_i = (C_i^l, C_{i1}^f, ..., C_{iM}^f, C_i^c)$ (the calculated end-to-end monetary cost for task t_i on each computation unit). See Eq. (4.14).

The optimisation problem, formulated as monetary cost (C) times energy (E), is based on the assumption that these elements contribute equally to the objective function:

- $E = e^T x$, where $e = (e_1, ..., e_N)$ represents the matrix of energy values for task t_i on all computation resources and e^T is the energy matrix transposition;
- $C = c^T x$, where $c = (c_1, ..., c_N)$ represents the matrix of cost values for task t_i on all computation resources and c^T is the cost matrix transposition;
- $x = (x_1, ..., x_N)$ (the tuple represents the decision variable of each task t_i); and
- N is the number of tasks. The optimisation function can be formulated as:

$$P_0 : min(E * C) \tag{4.19}$$

Subject to R_0

$$D_{t_i} < \partial_i, \forall t_i \in A$$
$$E < e$$
$$x_i^l, x_{ij}^f, x_i^c \in \{0, 1\}, \forall (i, j) \in NxM$$

where constraints of deadline $D_{t_i} < \partial_i$ and user mobile device energy $E < e$ should be satisfied.

P_0 is an NP-hard optimisation problem, due to its MINLP [110]. The resulting problem is a convex optimisation problem [112]. To prove the convexity of our problem, the binary decision variables $x_i^l, x_{ij}^f, x_i^c, \forall (i, j) \in NxM$ were relaxed into real numbers of range $[0, 1]$ (please refer to the work of Vu et al. [97] for the mathematical proof). Thus, the optimisation problem can be formulated as:

$$\widetilde{P}_0 : min(E * C) \tag{4.20}$$

subject to R_0 and:

$$(C3) : x_i^l + \sum_{j=1}^{M} x_{ij}^f + x_i^c = 1$$

$$x_i^l, x_{ij}^f, x_i^c \in [0,1], \forall (i,j) \in NxM$$

The BB algorithm is a commonly used search optimisation algorithm for solving ILP and MILP problems. The algorithm solves linear programming (LP) relaxations using restricted ranges of possible values of the integer variables. It attempts to generate a sequence of updated bounds on the optimal objective function value. Like dynamic programming, BB is an intelligently structured search of the space of all feasible solutions [113]. In BB, an optimisation problem is considered as a search tree, in which every tree node represents the transition to a subproblem after fixing a binary variable. Subproblems have the same objective function, bounds and linear constraints as the original problem, but without integer constraints.

Algorithm 3 provides a high-level abstraction of the behaviour of the BB optimisation method. Algorithm 3 takes two inputs: an application A and available computation resources set R. The algorithm aims to find the optimal offloading decision for minimising the cost.energy objective value. The optimisation objective $optVal$ is initialised to infinity because a minimisation problem is targeted. Algorithm 3 is a high-level description of how BB works, with the following steps.

1. The algorithm starts with an initial subproblem P_0.

2. Calculate the objective value $solObjValue$ of the stack top solution $toCheckSol$. In Lines 11-12, the solution (decision) is evaluated by calling $callSolObjectiveValue$ in Algorithm 4, and the optimum value is updated if a new minimum is found, Lines 16-18. A solution $toCheckSol$ represents the binary encoding for application A tasks allocation on system resources R. Only one position is assigned to 1, which indicates the target machine for that task. Based on solution $toCheckSol$, optimisation parameters, that is, execution time D_i, energy consumption E_i and monetary cost C_i are calculated via $callSolObjectiveValue$ in Algorithm 4.

3. Check to update the best solution $bestSol$.

4. Branch on $toCheckSol$ to produce new solutions (nodes), Line 20. The branching step is taken heuristically, according to one of several rules. Each rule is based on the idea of splitting a problem by restricting one variable to be less than or equal to an integer $J = 1$, or greater than or equal to $J+1$. On each branching step, two subproblems $toAddSubProblems$ are constructed by changing on one decision variable x_i. Technically, a subproblem is a candidate offloading decision.

5. Each subproblem is checked for problem upper and lower bound constraints using the function $checkIntegerConstraints$. This is called bounding a solution, Line 22.

6. Loop until the solutions stack ($subP$) is empty, Line 10.

7. Algorithm 3 returns the optimised application scheduling plan s and the minimum problem value $optVal$, Line 29.

Algorithm 4 provides the steps to calculate the objective value for an offloading solution sol. The algorithm takes three inputs: proposed solution sol, the application A and the computation resources R. The resource handler is responsible for collecting data about the current status of the resources R. Prior to running the MILP solver, Line 19, the algorithm starts with building the problem. A MILP problem has two main components. The first is the list of $Constraints$. Two constraints are included: the deadline constraint, Line 14, and the maximum available energy constraint, Line 18. A solution cost value is calculated at the cost estimator, which communicates with the queuing estimator to estimate task waiting time t_i at resource r_j. Moreover, the cost estimator collects data about mobile device energy status by communicating with the application profiler. To set the deadline constraint, the execution time for each task is calculated by calling the function $findTime$ and using Eq. (4.3), Line 11. The energy constraint expresses the cumulative energy consumed, Lines 12 and 18, using Eq. (4.10). The monetary cost is calculated for each task, Line 13, using Eq. (4.14) and the cost value for each

Algorithm 3 Find optimal application tasks schedule

1: **Inputs:**
2: Application tasks $A = \{t_i, ..., t_n\}$
3: Computation resources $R = \{r^l, (r_1^f, ..., r_m^f),\ r_1^c\}$
4: **Output:**
5: optimised application tasks allocation plan P_0
6: **Initialise:**
7: $optVal = \infty$
8: $bestSol = \{\}$
9: $subP = \{P_0\}$
10: **while** $Len(subP > 0)$ **do**
11: $toChecksol = subP[0]$
12: $solObjValue = callSolObjectiveValue(A, R, toChecksol)$
13: **if** $solObjValue > optVal$ **then**
14: $subP.removeAt(0)$
15: **else**
16: **if** $solObjValue < optVal$ **then**
17: $bestSol = toChecksol$
18: $optVal = solObjValue$
19: **else**
20: $toAddSubProblems = Branch(subP[0])$
21: **for** $i = 1$ to $Len(toAddSubProblems)$ **do**
22: **if** $checkIntegerConstraints(toAddSubProblems[i]) == True$ **then**
23: $subP.insertAt(0, toAddSubProblems[i])$
24: **end if**
25: **end for**
26: **end if**
27: **end if**
28: **end while**
29: RETURN $s, optVal$

task is added to the list, Line 15. Line 15 shows the model objective function of energy E and cost C. The MILP solver function, Line 19, uses constraints and cost matrices to run the solver to find the offloading decision with minimum cost value. Finally, the offloading decision and cost value is returned to Algorithm 3.

4.6 Performance Evaluation

The performance evaluation for the proposed multi-objective offloading technique was conducted using two experiments. The first was a real experiment to validate the model by comparing the results of the proposed execution model and offloading technique against real executions. For this experiment, a real MECC environment

was implemented, as illustrated in Figure 4.1, while in the second experiment synthetic application data was used to evaluate the proposed offloading technique.

4.6.1 Experimental Setup

The research objective was to study the contribution of parameters, such as the mobile network interface, task input data size distribution and number of application users, to the offloading optimisation objectives (monetary cost and device energy consumption). Firstly, a mobile network interface determines the cellular

Algorithm 4 callSolObjectiveValue

Inputs:
Application tasks $A = \{t_i, ..., t_n\}$
Computation Resources $R = \{r^l, (r_1^f, ..., r_m^f),\ r_1^c\}$
Problem solution - Decision Variable $X = \{x_i, ..., x_n\}$
Output:
MILP Problem Parameters
Begin:
$Constraints = \{\}$
for $i = 1$ to $Len(A)$ **do**
  for $j = 1$ to $Len(R)$ **do**
    $T[i,j] = findTime(t_i, r_j)$
    $E[i,j] = findEnergy(t_i, r_j)$
    $C[i,j] = findCost(t_i, r_j)$
    $Constraints[i,j] = AddToConstraint(T[i,j], t_i.deadline)$
    $CostVal[i,j] = AddToObjective(E[i,j] * C[i,j])$
  end for
end for
$Constraints[Len(A), Len(R)] = AddToConstraint(Sum(E), maxE)$
$X, ObjVal = SolveProblem(CostVal, Const)$
Return $X, ObjVal$

Table 4.2: Network interface bandwidth

Network Type	**Min. Bandwidth (MB/s)**	**Max. Bandwidth (MB/s)**
3G	2	5
4G	8	12
WiFi	25	30
Latency	**Min. Latency (s)**	**Max. Latency (s)**
	0.85	6.5

data transfer technology that is available to the user device to send or receive data. Table 4.2 provides details about the bandwidth distribution of each mobile network interface and the average data transmission latency over these interfaces. Secondly, the application size is initialised with 30 tasks. This number of tasks provides the ability to verify a convenient range of data size, for the purposes of a data-intensive application, without complicating the application structure. Thirdly, Table 4.3 shows the data size distributions applied in the experiment. Finally, for task setup, the same setup provided in the previous chapter, section **??**, was used. The edge computation cost is assumed to be 40% of the cloud compu-

TABLE 4.3: Task input data size distribution

Data Distribution	**Min. Size (MB)**	**Max. Size (MB)**
Small	20	200
Medium	200	500
Large	500	2000
X-Large	2000	4000

tation cost. In addition, it is assumed that CPU cores have the same clock speed and MIPS, and that the computation machine has enough memory at the time of processing. Cost for the edge nodes can be specified by the owner or resource provider. Table 4.4 provides details about the computation resources used in the experiment.

TABLE 4.4: Experiment resources configuration

Resource Name	**#Cores**	**#Nodes**	**Computation Cost ($/Hour)**
Mobile Device	2	1	0.001
Edge Node	2	8	0.0742
Cloud Server	32	1	0.3712

4.6.2 System Profiling

System profiling followed the same procedure provided in the previous chapter, section (3.5.2). The profiling process includes network bandwidth and latency profiling to examine bandwidth boundaries for data communication between computation and storage units at cloud and edge layers.

A profiling process to compute an approximation of energy and bandwidth estimation parameters was implemented. Prior to a profiling iteration, an experiment configuration is obtained from data size distribution. For each configuration setup, a BoT application of 30 tasks is executed, and averages of processing, communication and waiting energy values are recorded for 30 executions. PowerTutor software is used for power estimation and provides energy consumption estimates within 5% of actual values [105].

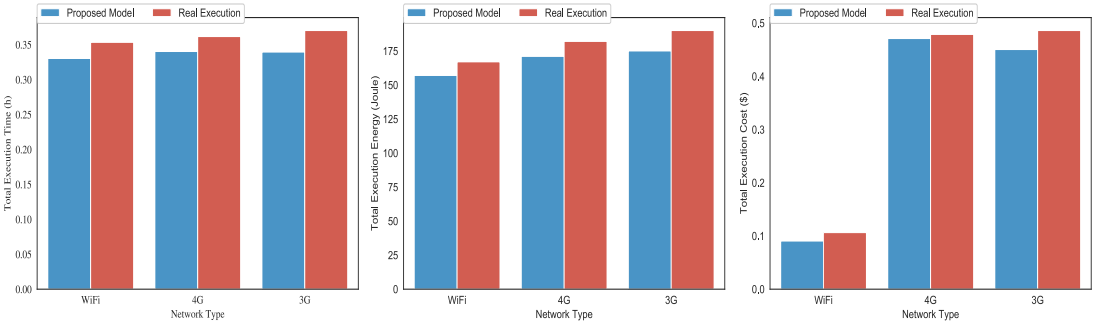
The profiling process includes network bandwidth and latency profiling to examine bandwidth boundaries for data communication between computation and storage units at cloud and edge layers. Energy parameters are used to calculate the energy the device consumes for task processing, data transfer and device waiting for external task processing ${\epsilon_i}^P$, ϵ^C, ϵ^W, respectively. The profiling process considers the parameters of task data size, mobile network, mobile device processing capability and physical data file transfer between data storage and computation locations. The experiment was repeated for each combination of data size and bandwidth. Table 4.3 shows the data size distributions for data files used in the profiling experiments. Also, as part of system profiling, the data sensitivity factor ω_i for a task t_i was estimated. The sensitivity factor measures how task processing is sensitive to the change in task data size. Finally, Table 4.2 shows bandwidth profiling results.

4.6.3 Model Validation

This section describes validation for the proposed execution model using a real execution scenario. The validation objective was to evaluate optimiser stability and validity by comparing results obtained from the optimisation model with those collected from the real experiments. Results include execution time, consumed energy and monetary cost of computation and data communication.

Evaluation experiments were run with 30 task applications, small input data size (see Table 4.3) and three mobile network interfaces (3G, 4G and WiFi). Figure 4.2

shows the evaluation results of the three model parameters based on the change in the mobile network interface. The experiments predict slight estimation errors for three model parameters, with 7%, 7% and 12% for execution time, consumed energy and monetary cost, respectively. On average, the estimation error per scenario is around 9%.

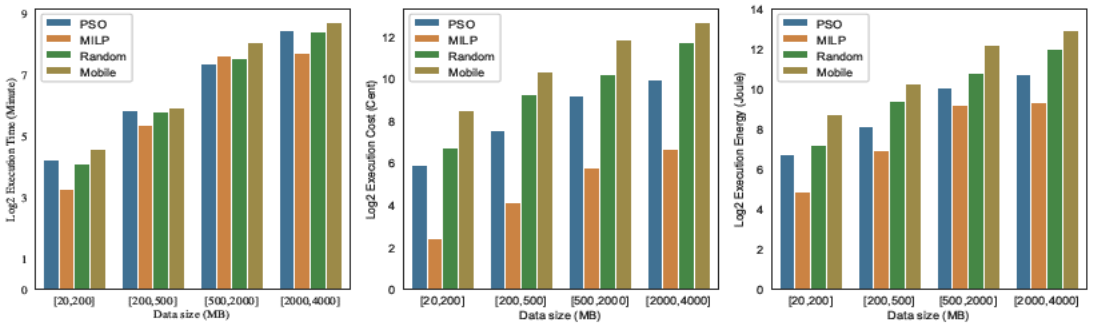


(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 4.2: Evaluation of proposed optimiser with real scenarios for three different communication networks

4.6.4 System Evaluation Results

After validation of the execution model, similar setup was used to evaluate the proposed offloading technique under different working conditions. The procedure began by comparing the proposed optimisation technique with PSO, mobile and random techniques. The PSO technique was employed on the previous chapter in data-intensive mobile application offloading [114]. The mobile technique involves executing all application tasks in the user device. The random technique is a baseline technique of much less complexity than the proposed techniques. The aim was to learn how the proposed techniques perform against the baseline and measure the performance gap. The task deadline is a soft deadline, and applications can be still completed with the random technique (with low QoS achievement). Experimental results provide insight on how the optimiser responds to a variety of parameters, such as data size, bandwidth, deadline and number of users, when constructing an optimised application offloading plan.

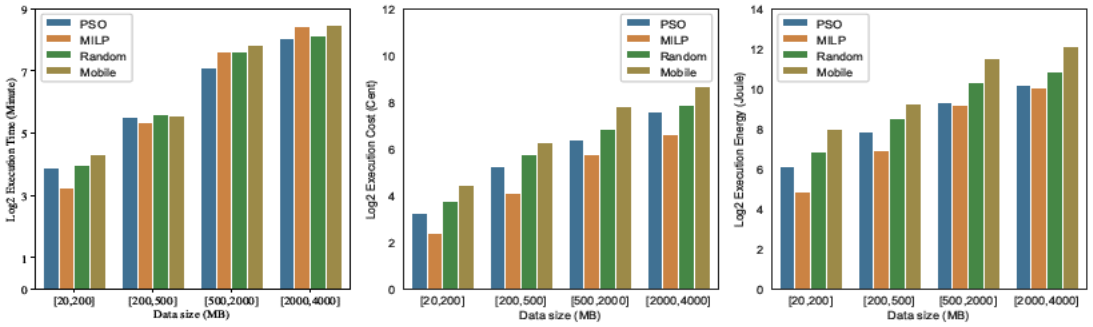


(a) Execution time (b) Monetary cost (c) Mobile energy

FIGURE 4.3: System performance measurements with 4G network interface for different data sizes

Figures 4.3 and 4.4 compare the results of applying the four techniques for optimising model parameters with respect to the variation in task input size. With the 4G network interface, Figure 4.3 shows that all techniques demonstrate a linear increase in execution time, monetary cost and mobile energy consumption as data size increases. The MILP optimiser produces the lowest scores of energy and cost; there is a large performance gap between it and the second-best optimiser, the PSO, even for large input files, proving the stability of the model. With the WiFi network, as shown in Figure 4.4, the MILP technique minimised the execution cost regardless of the change in input data size. This can be inferred from execution time change behaviour, in which the MILP tried to work with maximum task deadlines to schedule tasks in resources with minimum cost, in this case, edge resources. Thus, in some cases, the MILP technique does not provide the least execution time among other techniques. This happens because the execution time is a constraint, not an optimisation objective of the algorithms. So, rather than trying to minimize the execution time, the algorithms try to guarantee that applications complete by their deadlines. Overall, results revealed that the proposed technique reduced the execution cost for data-intensive applications by an average of 46% and 76%, in comparison to PSO and full execution on a mobile device, respectively. In addition, the model provides energy reductions of 35% and 84%, respectively.

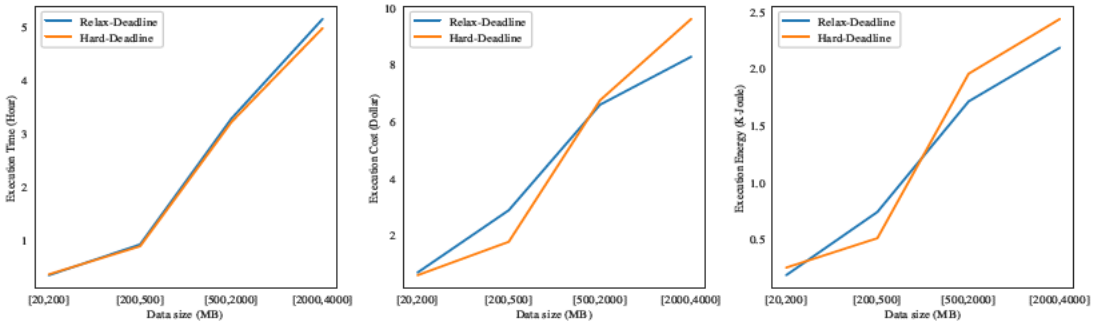
To study the contribution of the task deadline, two scenarios were applied: a relaxed deadline and hard deadline. To construct the two deadline models, the



(a) Execution time (b) Monetary cost (c) Mobile energy

FIGURE 4.4: System performance measurements with WiFi network interface for different data sizes

execution model proposed by Anglano and Canonico was applied [104]. The model used to determine the task deadline is based on the level of granularity. A relaxed deadline model is assumed to have twice the granularity of a hard deadline model. Application task deadline values will be uniformly distributed in the interval $[X \pm 0.5X]$, where X is the granularity of the tasks. Figure 4.5 compares the two scenarios with respect to the change in input data size. It can be concluded that with large input data the optimiser works toward scheduling tasks in high-performance computation machines in a public cloud, which explains the cost difference in the two scenarios. In addition, the energy consumption is increased, reflecting the greater mobile waiting time for remote execution.



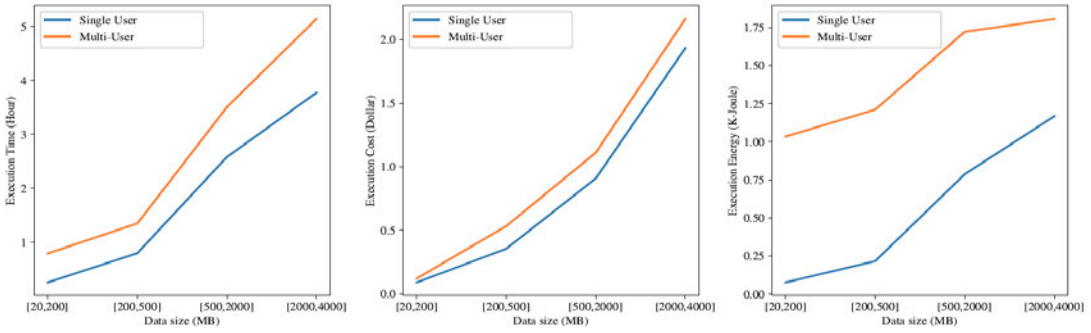
(a) Execution time (b) Monetary cost (c) Mobile energy

FIGURE 4.5: The impact of deadline sensitivity on optimisation parameters

The number of users parameter was added to the experiment to study the impact of task waiting time on energy consumption and total monetary cost. A task waiting time is estimated based on the G/G/1 queuing model [108]. It is assumed

that both cloud and edge servers behave as G/G/1 queuing systems with variation in task arrival rate and service time. The arrival rate represents the number of received tasks in a certain time slot and the service time measures the time required to process the incoming task. However, application tasks differ in computation requirements; thus, the queuing system needs to handle heterogeneity in service times. To overcome this issue, we performed an experiment to profile queuing results at cloud and edge servers and used the means of variable rates and service times to feed the optimisation model. Two scenarios, single-user and multi-user, were tested: single-user means only one user application is in execution, while in the multi-user scenario many of these applications are submitted for execution.

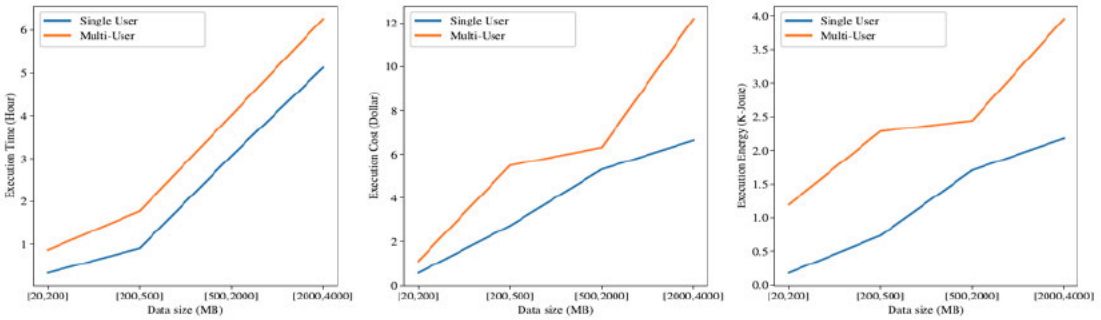
Figures 4.6 and 4.7 show that the difference in execution time between multi-user and single-user scenarios is increased marginally with greater input data size, due to the increase in application waiting time. With the WiFi network, as shown in Figure 4.6, the optimiser was able to optimise the cost by allowing high waiting time and thus, high energy. On the other hand, with the high-cost network, such as 4G as shown in Figure 4.7, the time difference between the two scenarios remained steady, allowing energy savings.



(a) Execution time (b) Monetary cost (c) Mobile energy

FIGURE 4.6: The impact of multi-user and single-user scenarios on system performance: WiFi mobile network

Figure 4.8 compares the optimisation time of the proposed MILP-based technique and our previous PSO-based technique [114]. The figure confirms how PSO-based optimisation time increases exponentially as the number of tasks increases (application size). The MILP technique shows near-zero optimisation time variation with increase in application size, while PSO demonstrates a 45% time increase on



(a) Execution time (b) Monetary cost (c) Mobile energy

FIGURE 4.7: The impact of multi-user and single-user scenarios on system performance: 4G mobile network

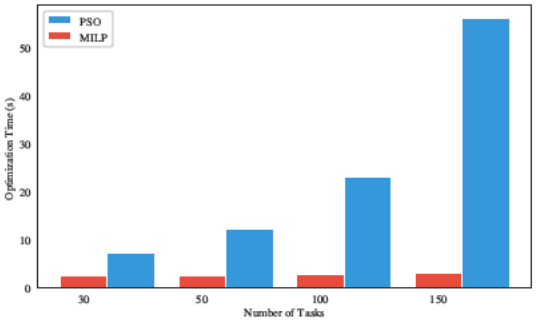


FIGURE 4.8: The optimisation times for MILP and PSO

average. For example, running applications of 100 and 150 tasks using the MILP technique will take 2.7 and 2.9 seconds, respectively, while running the same application using PSO will take 23 and 56 seconds, respectively. This result reveals that the proposed technique has a low time complexity and can be adapted for large-scale data-intensive applications.

4.7 Summary

We proposed a mobile application offloading algorithm to schedule data-intensive mobile applications in the MECC environment. The algorithm generates an application execution plan which accounts for application size, input data size, available network bandwidth and mobile device energy. Results showed the ability of the MILP-based technique to reduce the execution cost and energy consumption linearly with change in data size. For latency-sensitive application models, the

results imply that data size is the major driver of both monetary cost and device energy, with the mobile network model making a smaller contribution. Moreover, the work measured the impact of the number of active application users on model parameters. Results demonstrated the model's ability to handle data-intensive applications by improving the efficiency of MECC. Moreover, the MILP-based model has significantly lower computation time than the PSO technique. Finally, the adoption of MECC demonstrated high viability with respect to reducing energy consumption and the monetary cost.

However, so far, only the offloading optimisation in respect to the BoT application model has been studied. The next chapter describes experimental work related to data-intensive application offloading planning and optimisation for other application models, such as workflow and stream models. In addition, other optimisation objectives, such as energy-only and cost-only, are studied.

Chapter 5

Performance Analysis of Mobile, Edge, and Cloud Computing Platforms for Distributed Applications

Mobile devices and their corresponding services experienced a tremendous expansion of in almost every social and business in human life. Mobile services cover wide range of applications for collaboration, communication, monitoring, tracking, streaming, and many others. This engagement brings significant challenges for mobile capability to effectively contribute on computation cycles due to limited computation power, the dependency on short-term energy power, and the sensitivity to transmission network. A common technique to resolve mobile shortcomings is to offload complex computations to more powerful resources such as edges, clouds, mobile clouds or integration between them. With the variation of mobile applications, computation offloading becomes a complicated task to align the unique characteristics and user QoS requirements for each application to a convenient offloading plan. The availability of powerful resources at different computing layer is also another challenge for offloading techniques. A question is how a user can select a mobile-aware computing paradigm for a specific application,

such that, it satisfies QoS requirements, considers the application structure and meets the challenge of handling large input data volume. The idea of this chapter is to assess the performance of mobile-aware computation systems in running different types of application models through a quantitative analysis of measurement parameters including energy consumption and monetary cost. The following sections describe comprehensive analysis undertaken to answer the question as well as provide recommendations regarding offloading data-intensive mobile applications on.

5.1 Introduction

Chapters 3 and 4 proposed algorithms and techniques for data-intensive application offloading on hybrid-MCC and MECC systems, respectively. This chapter extends that work by applying these techniques to multiple application models. The chapter provides a comprehensive analysis of many aspects of mobile-related computation models. Its objective is to provide recommendations for selecting a mobile-aware computation paradigm to execute data-intensive mobile applications based on aspects such as application model, data size, mobile network performance, and mobile energy status. The selection of a computation model is based on the amount of energy consumption and the total monetary cost of executing application tasks and transferring data between computation nodes.

The variation in mobile usage in many application contexts means different types of application models must be accommodated. This chapter considers the features of three models, namely, BoT, workflows and IoT. These features determine the dependency between application tasks in terms of computation and sharing data. Application complexity and structure should be considered while planning for offloading to a mobile-aware computing environment. In combination with the size of the application data, and the quality of the communication network, the application structure adds another dimension to the offloading optimisation process. This chapter presents a comprehensive analysis that shows how to select

the best mobile-aware computing system to offload a data-intensive application based on parameters of data size, network quality and application structure. The motivation of this work was to reveal how these parameters could affect an application scheduler's selection of mobile-aware computation environments, that is MC, MCC or MECC. The performance assessment of these computation environments will be based on a quantitative analysis of their ability to run different types of application models such as BoT, workflows and IoT applications while meeting QoS and optimisation measurements including deadline, energy consumption and monetary cost.

Each computing model has its own unique characteristics and is appropriate for specific types of mobile applications. For example, MC reduces computation cost, MEC highly recommended for time-sensitive applications, MCC can support computation-intensive and data-intensive applications with powerful computation capacity, and MECC integrates MEC and MCC to resolve the emergent direction scope of data analytics and IoT-based applications. Furthermore, user QoS requirements are varied, ranging from saving mobile energy, reducing computation cost, and minimising application execution delay. The question is how a user can select a mobile-aware computing paradigm for a specific application such that it satisfies QoS requirements and considers the application structure and contextual execution environment. Moreover, this chapter was designed to generate insights into ways the mobile communications industry could realise cost savings and high-quality data-aware offloading solutions by adopting new technologies such as edge computing and region-based local networks.

The chapter is structured as follows. Section 5.2 provides an overview of computing environments. Section 5.3 describes the system model, offloading technique and cost models, while experimental work is presented in Section 5.4. Lastly, Section 5.6 illustrates the chapter summary and main findings.

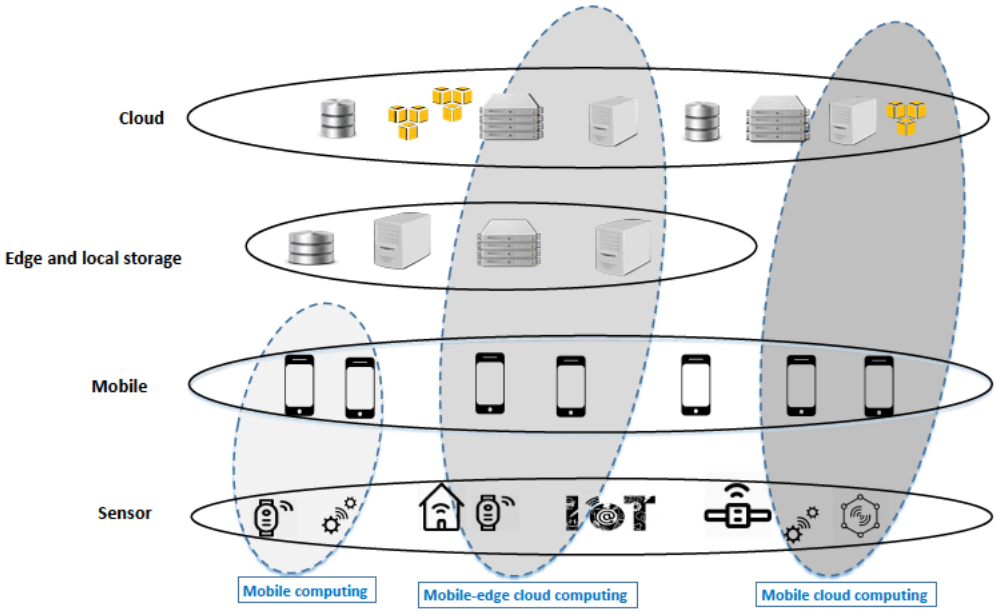


FIGURE 5.1: Mobile-edge cloud computing architecture

5.2 Overview of Cloud, Edge, and Mobile Environments

This section provides an overview of the adopted computing architecture. Figure 5.1 shows three levels of computation environment. At each level, different types of resources are integrated to processes mobile application tasks. It is assumed that data can be generated from fixed data storage (local edge or cloud), or from IoT devices like sensors. IoT data are included to study the impact of data transfer on the offloading optimisation decision. Mobile devices are the core computing units for mobile-aware computing models; they host the application and act as a computation unit to run application tasks. In MC, all the computation for an application, including task processing and data transfer, is handled by mobile resources. Thus, it is challenging for mobile devices to deliver complex high-performance functionalities at the lowest possible energy level. However, as discussed already, mobile device I/O processing and network communications are energy-hungry components. In addition, even though mobile devices are equipped with high-performance computation resources, they still encounter significant challenges in meeting the requirements of computation-intensive and data-intensive

mobile applications.

To avoid these shortcomings of computation and storage, cloud resources are integrated. The cloud layer offers high accessibility with pay-as-you-go computation resources. However, the large distance between a mobile device and the cloud server introduces higher latency in cloud computing than edge computing. In MCC architecture, a mobile device is able to offload heavy workload to the cloud, benefiting from its high computation capabilities to reduce the overhead of running these tasks locally in the user's mobile device.

Mobile cloud computing architecture has limited ability to handle the migration of high-volume files due to the long distance between data sources and cloud servers, which may cause high data transfer latency and incur additional monetary cost. Edge computing can resolve these issues as it offers computation and storage resources closer to mobile devices and data sources [30]. In addition, edge computing allows time access for network performance data related to latency and bandwidth, and be implemented in many application scenarios for real time and interactive applications [37].

5.3 System Model

This section presents an overview of the optimisation technique. It includes a description of types of application models, the application modelling and abstraction, the cost models and the optimisation technique.

5.3.1 Application Model

An application model represents the internal structure of the application tasks and how they are related in terms of computation and data dependency. This chapter presents work on three types of application models, namely, BoT, workflow and IoT. Figure 5.2 shows an abstraction of these models. Figure 5.2 (a) shows the BoT, which is a task-independent application model where tasks are fully isolated

in their input data and computation logic [100]. This model is expected to have additional overheads in terms of data transfer time and cost. The independence of tasks increases the opportunity of transferring and processing large amount of data.

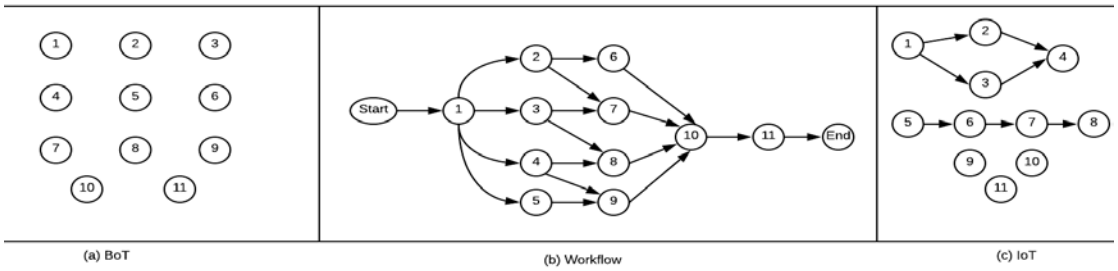


FIGURE 5.2: Application models abstraction

In contrast, in the workflow model, Figure 5.2 (b), tasks are dependent; they transfer their processing outcomes to corresponding tasks based on the workflow structure. The dependent structure of a workflow model has a significant impact on optimising application execution, because it determines the data flow and the amount of data to be passed between tasks. It is convenient to construct workflow execution schedules that reduce the time and cost overheads of transferring data between dependent tasks. A workflow is a common application model representation, and is adopted in a wide range of application domains, such as scientific domains, stream processing and data analysis [115]. Lastly, Figure 5.2 (c) shows IoT applications, in which data is collected from IoT devices (such as sensors) either in online or offline mode, as is common in application domains like data analytics and real-time monitoring [116]. This chapter describes an investigation of the contribution of the IoT data collection stage to application offloading optimisation. It is assumed that an IoT application is a combination of of BoT and workflow models.

The application modelling includes high-level task modelling to reflect the three applications models (BoT, workflow and IoT). Table 5.1 describes the mathematical notations used in application modelling. A data-intensive mobile application A represents the execution of dependent or independent set of tasks, which is

TABLE 5.1: Problem modelling notation

Symbol	**Definition**
t_i	Application task i
L_i	Task input file location, either locally or remotely
s_i	Task input size
w_i	The number of task execution instructions
∂_i	Task deadline

modelled as:

$$A = \{t_1, t_2, \ldots, t_n\} \tag{5.1}$$

where n is the number of tasks. A task t_i is modelled as:

$$t_i = \{L_i, s_i, w_i, \partial_i\} \tag{5.2}$$

5.3.2 Overview of Cost Models

Cost estimation models are involved in formulating the mobile application scheduling and allocation problem. To choose the application execution plan, task execution time, energy and monetary cost must be estimated. The task execution time for task t_i is the sum of task processing time D_i^P, data communication time D_i^C, and task average waiting time D_i^W for remote execution. Here, edge and public cloud computation resources are modelled as G/G/1 queues.

$$D_i = D_i^P + D_i^C + D_i^W \tag{5.3}$$

The energy consumed by mobile device E_i to execute t_i is estimated by the sum of the total processing energy E_i^P consumed by the mobile device, the waiting energy E_i^W, and E_i^C (the mobile energy consumed during data communication).

$$E_i = E_i^P + E_i^C + E_i^W \tag{5.4}$$

The monetary cost is the amount of money to run task t_i in the target computation environment P_r. This has two parts: total remote task processing cost C_i^P in edge nodes or the public cloud, and total data communication cost C_i^C.

$$C_i = C_i^P + C_i^C \tag{5.5}$$

The detailed modelling of task execution time, energy and monetary cost is provided in section 4.4.3, 4.4.4 and 4.4.5, respectively. The next section describes the proposed offloading technique, which employs the cost models to find an optimised application offloading plan.

5.3.3 Overview of the Optimisation Technique

The system objective is to find an optimal solution in which total consumed energy and total monetary cost are optimised with respect to task deadline and device energy. A solution is represented as a tuple of each task t_i and the selected computation environment. This work utilises MILP to address cost and energy optimisation for an MECC architecture. A MILP formulation essentially necessitates that all objective functions and constraint equations are linear.

The optimisation problem is formulated as monetary cost (C) times energy (E) which is based on the assumption that they contribute equally to the objective function. The BB algorithm is a commonly used search optimisation algorithm for solving ILP and MILP problems. The algorithm solves LP relaxations with restricted ranges of possible values of the integer variables. It attempts to generate a sequence of updated bounds on the optimal objective function value. Like dynamic programming, BB is an intelligently structured search of the space of all feasible solutions [113]. In BB, an optimisation problem is considered as a search tree, in which every tree node represents the transition to a subproblem after fixing a binary variable. Subproblems have the same objective function, bounds and linear constraints as the original problem, but without integer constraints.

5.4 Experiment for Data-Intensive Application Offloading

The main motivation of this chapter was to produce insights that would support users to select a cost-efficient offloading plan to run their applications in multiple computing environments. To achieve this, the work reported in this chapter included designing and implementing an experiment to provide insights on how an offloading decision can be obtained based on application model, data size, context parameters and computing environment. The experiment examined the optimisation decision with respect to the variation on the aforementioned parameters. In addition, the work described in this chapter aimed to evaluate the performance of various types of mobile application models in nominated mobile-aware computing environments. The rest of this section provides details about the evaluation metric, experiment configuration, and the main insights from the experimental results.

5.4.1 Evaluation metrics

A MILP technique was adopted to optimise the offloading and execution of a mobile application in a computing environment. The optimisation technique aims to construct an offloading plan to map application tasks on environment resources. An optimised scheduling plan reduces the energy consumption of the user's mobile device and the total monetary cost. Both optimisation parameters are affected by the processing time parameter, which includes task processing and data communication. The processing time is an optimisation constraint handled at task level and accordingly at application level.

Conserving energy is a critical aspect of mobile application optimisation; losing device energy is a single point of failure for running the device. In this chapter, three energy consumption operations are considered: task processing, data transfer and device waiting (or idle) for remote execution. The contribution of each

process is subject to many parameters, including data size, network bandwidth and application complexity. The monetary cost, which involves data transfer and processing cost. For data-intensive applications, data communication cost is not trivial, particularly when a cellular network is utilised. The optimisation algorithm is designed to handle the transfer of large data files to reduce the use of mobile network bandwidth. In addition, this work considers the impact of large data files in task processing cost. Moreover, the cost estimation depends on task complexity and how it responds to the change in data size. Here, energy and cost parameters are considered equally in the optimisation decision.

5.4.2 Experimental Setup

This section outlines the setup used to run data-intensive applications on multiple computing models. To recap, this chapter presents an investigation of the role of mobile application structure on offloading optimisation decisions, because it defines the computation and data dependency among application tasks. Offloading optimisation is examined with three different models, namely, BoT, workflow and IoT. In BoT, tasks are independent on computation and data. The BoT is convenient for studying offloading optimisation for data-intensive applications with data source distribution between local, edge and cloud storage units. The workflow model involves transferring data between tasks. Thus, the task computation target plays the role of data source for dependent tasks. In the IoT model, a combination of BoT and workflow models is assumed, and the impact of collecting data from IoT devices is studied. The data collection can be handled by the mobile device or by an edge node.

5.4.2.1 Computing Resources

The experiment was designed to investigate the efficiency of the computing model on the optimisation decision. Three models were studied: MC, MCC and MECC. Table 5.2 provides details about the resources configuration employed. A small

fraction of cost for running a task on a mobile device is assumed. Processing cost on edge was assumed to be 40% of the cost on cloud.

5.4.2.2 Workload Model

In this chapter, different workload models are adopted for BoT and workflow application models. This section provides the details of workload and data models used in the experiments.

The Montage workload model provided by Bharathi et al. [117] was employed. A Montage-like workflow is generated to handle sky image processing. The workflow includes a set of tasks to import images, find differences, fit and concatenate, and finally create the mosaic. Figure 5.3 shows the Montage workflow structure. Montage workload model is basically designed to handle complex and heavy computations which are hosted in powerful computing systems like public clouds. This type of workload may not be the ideal for edge-based systems. However, the idea of adopting Montage is for it's workflow-based representation to asses the behavior of proposed offloading techniques on various computing systems. In the workflow model, the computation complexity depends on the number of images and the level of overlap between images. For the purpose of offloading optimisation, the attributes of data sensitivity must be set for each application task. The data sensitivity factor relates the data size with computation time.

The provided profiling data was used to extract the parameters of data sensitivity and generated output sensitivity. Table 5.3 shows the data and output sensitivity values for workflow task types. Moreover, to have variation in data size, for the purpose of studying data change contribution on offloading optimisation decision,

TABLE 5.2: Experiment resources configuration

Resource Name	**#Cores**	**Memory (GB)**	**Cost $/Hour**
Mobile: LG Nexus 5	4	2	0.001
Edge: Machine Intel Xeron E5-1630	16	4	0.074
Cloud: Intel Xeron E5-2686	32	16	0.371

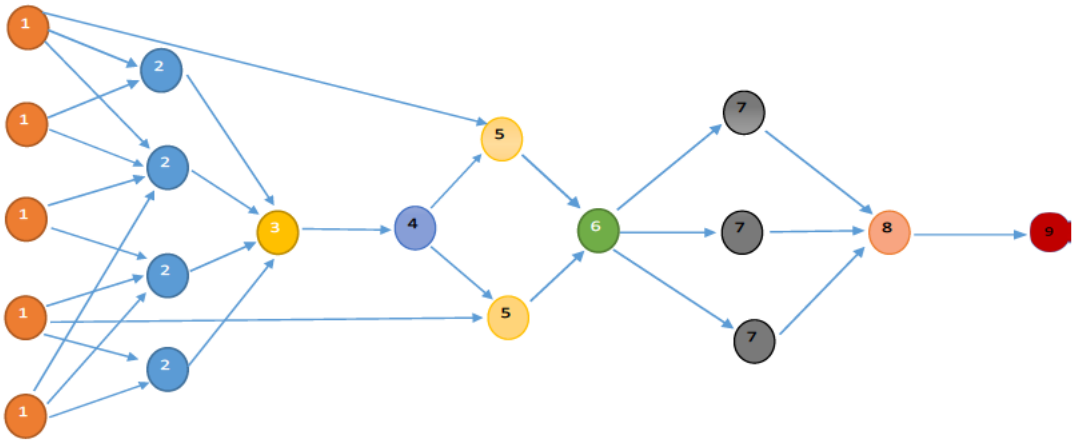


FIGURE 5.3: Montage Workflow

TABLE 5.3: Workflow sensitivity factors

Task	Data Ssensitivity	Output sensitivity
mProjectPP	0.725	8.955
mDiffFit	0.095	0.054
mConcatFit	2.250	1
mBgModel	1.808	0
mBackground	0.288	1
mImgTbl	0.287	1
mAdd	0.815	0.926
mShrink	0.356	0.020
mJPEG	15.894	0.048

a uniform distribution for image size and number of input images was adopted. Table 5.4 provides the details of the data workload model.

For the BoT model, a task workload model considering computation complexity, CPU load and data sensitivity was developed. The task complexity is determined by the number of MIPS instructions and the percentage of CPU usage for processing. The data sensitivity measures the contribution of data change on task complexity or execution time. The data size distribution provided in Table 5.4 was used. Table 5.5 shows two task complexity models for low and high-computation applications.

For the IoT model, the workflow and BoT workload models were integrated. An IoT application was designed as mini-batches of workflows and a BoT application.

TABLE 5.4: Data size distributions

Number of images [min, max]	**Data size [min, max] (MB)**
[1, 10]	[5,50]
[10, 20]	[50,100]
[20, 100]	[100,500]
[100, 200]	[500,100]
[200, 400]	[1000,2000]
[400, 600]	[2000, 3000]
[600, 800]	[3000, 4000]
Image Size Distribution	[3.9, 5.2]

TABLE 5.5: Task complexity models

Task Complexity (MIPS)	**Low**: [20 - 100] **High**: [200 - 700]
CPU Load (%)	[0.1 - 0.9]
Data Sensitivity (%)	[0.2 - 0.8]

5.4.2.3 Network Model

For data-intensive applications, the quality of mobile network has a significant influence on the transfer time. Three types of networks, WiFi, 4G and 3G were tested. Table 5.6 presents the network model applied in this experiment. The minimum and maximum bandwidth values of each network were based on profiling from previous work [118] the minimum and maximum bandwidth values of each network were set.

TABLE 5.6: Network interface bandwidth

Network Type	**Bandwidth (MB/s) [Min, Max]**	**Cost ($/GB)**
3G	[2,5]	1.0
4G	[8,12]	1.0
WiFi	[25,30]	0.05
Latency	**Min. Latency (s)**	**Max. Latency (s)**
	0.85	6.5

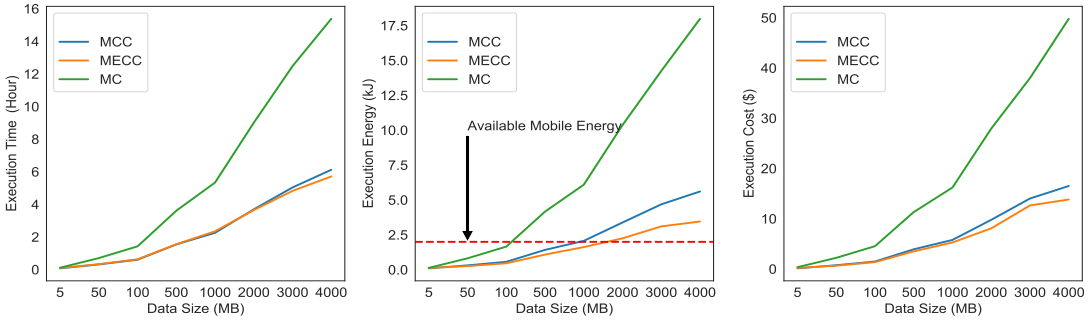
5.4.3 Performance Evaluation

This section discusses the experimental results of running the offloading optimiser with the experimental setup. For each application model, the impact of variation of data size and bandwidth values is highlighted.

5.4.3.1 BoT application model

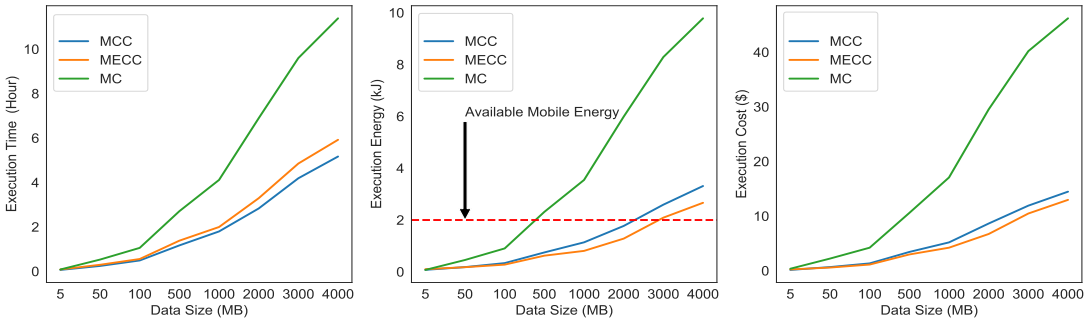
A BoT application execution involves running tasks in a separate mode where tasks are independent in terms of data and computation. In the context of data-intensive applications, this behaviour determines the requirements for transferring large files over the mobile network.

Figure 5.4 shows the result of running the BoT application over a 3G network. With 3G, the optimiser will try to find an optimised offloading decision to meet the task deadline and reduce the incurred cost of data transfer over the cellular data network. MCC and MECC demonstrate similar behaviour for application execution time along all data size variation intervals. With MECC, the optimiser reduces the energy consumption and cost compared to MCC, particularly with medium and large data size input files. This is due to the ability to offload heavy tasks to nearby edges, and thus reduce the energy consumption for local



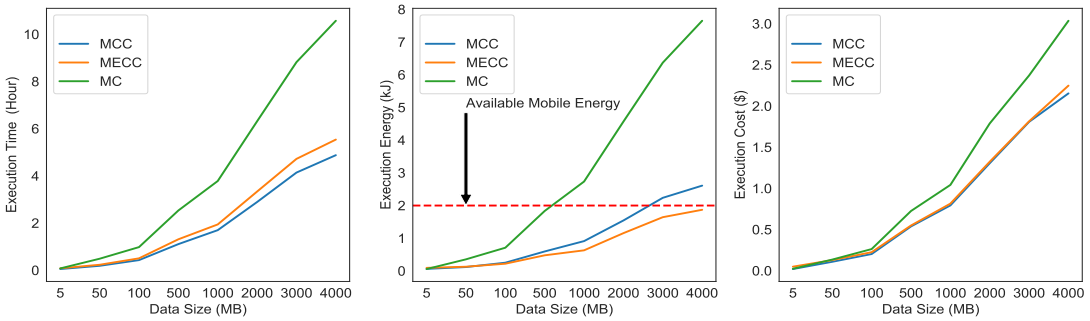
(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.4: BoT application model: 3G network



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.5: BoT application model: 4G network



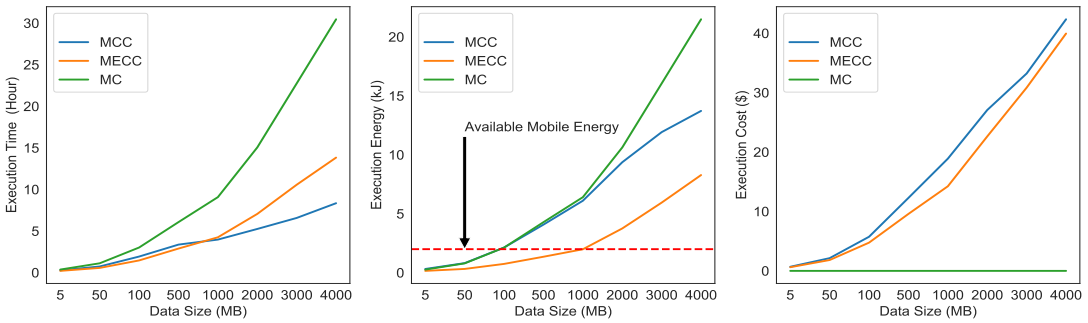
(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.6: BoT application model: WiFi network

processing and device waiting. However, Figure 5.5 demonstrates the contribution of higher bandwidth, using a 4G network, to reducing energy consumption. Moreover, with a 4G network, the optimiser tends to perform more computation using cloud resources to benefit from their high capability, as well as low waiting time compared to edge resources. This opportunity is also applicable in the case of WiFi network availability. Figure 5.6 provides the result for the WiFi network case. The three computation environments offer convenient BoT cost optimisation and reduce MECC costs with large data files. Even though the results confirm the limitations of MC due to energy shortage, the adoption of an external mobile energy source would be an effective enabler of local execution for data-intensive mobile applications. Figures 5.5 and 5.6 show that the mobile device is capable of running BoT applications even with medium data size on high-bandwidth networks.

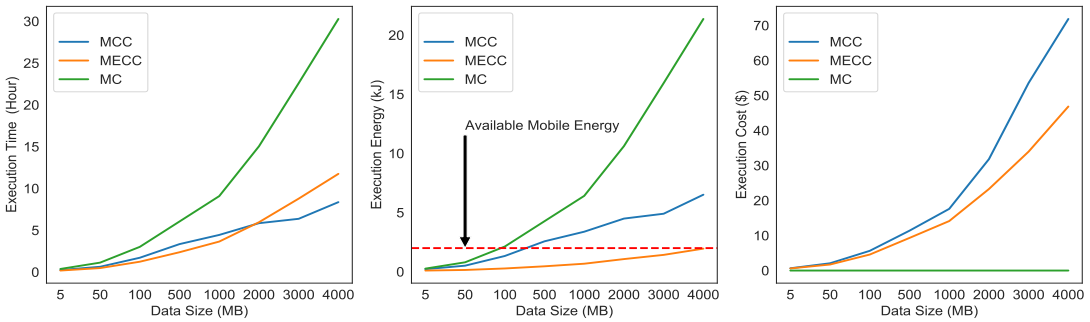
5.4.4 Workflow application model

A workflow application is a computation model in which application tasks are dependent. In workflow execution, the data location for the first task has a significant impact on the overall optimisation process. Here, the experiment was run with randomisation for the first task to obtain a convenient and stable offloading decision. This section details an analysis of running a workflow application and an investigation of the impact of mobile network and task input data size on evaluation parameters.



(a) Execution time (b) Mobile energy (c) Monetary cost

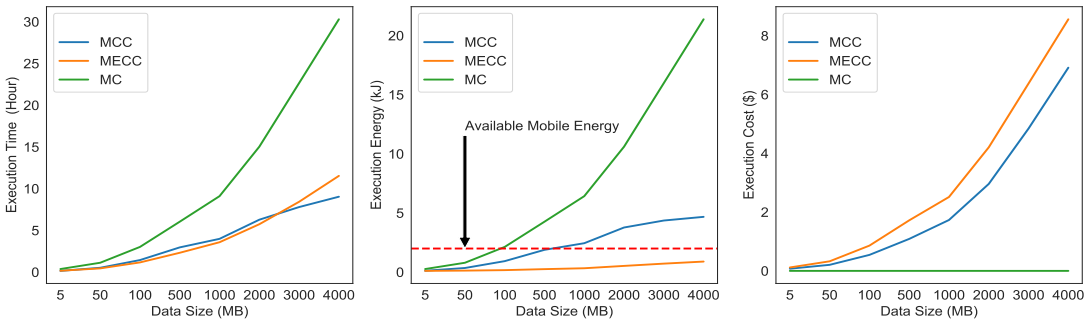
FIGURE 5.7: Workflow application model: 3G network



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.8: Workflow application model: 4G network

Figures 5.7, 5.8 and 5.9 show the results of running the workflow application with 3G, 4G and WiFi mobile networks, respectively. An interesting result is the advantage of MCC over MECC in minimising the workflow execution time, even though results show that MECC offers a substantial energy saving compared to MCC and MC. With the MCC model, the offloading optimiser tends to run dependent tasks on cloud VMs which efficiently reduces the high transfer time of large data files. With the MECC, the offloading optimiser processes some workflow tasks at the edge layer to reduce computation cost, but this may lead to increase in overall execution time due to data migration between edge and cloud layers. In addition, with the low-bandwidth network (3G) in Figure 5.7, the incurred cost gap between MECC and MCC reduces as the data size increases. To overcome the limitation of 3G network bandwidth in transferring large data sizes, an optimised solution would be offloading workflow tasks to the cloud. On the other hand, with 4G, in Figure 5.8, the high bandwidth allows an efficient joint computation



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.9: Workflow application model: WiFi network

between edge and cloud resources to reduce the computation cost. In the case of the WiFi network, Figure 5.9 shows that MCC provides greater savings than MECC. This is due to the efficient usage of the high bandwidth and low-cost WiFi network in mobile-cloud data transfer. This behaviour supports the conclusions in the previous section about running the BoT with a WiFi network.

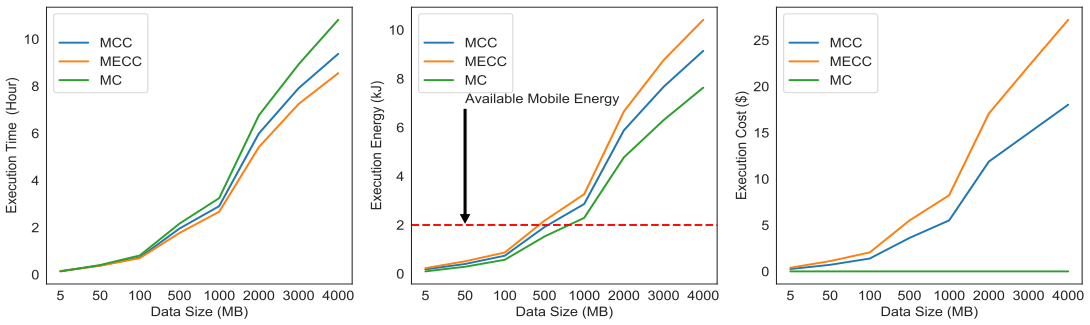
5.4.5 IoT application model

As explained earlier, IoT applications are common data analysis applications in which data is collected from IoT devices (such as sensors) either in online or offline mode. An IoT application was modelled as a combination of BoT and workflow sub-applications. In addition, for IoT offloading optimisation, the contribution of data collection stage was studied. Application data is collected by the user's mobile or by a stationary edge device. This section provides an analysis of the execution of an IoT application based on various network bandwidth and data size conditions in the three computing environments.

Figures 5.10, 5.11 and 5.12 present the experimental results of an IoT application execution when the user's mobile device is the data collection instrument. Results show that for all network types, the three computing paradigms provide similar results for application execution time. With the 3G network, in Figure 5.10, MC demonstrates high ability to optimise energy consumption and monetary cost. Thus, local processing is a preferred option when data is collected locally and a

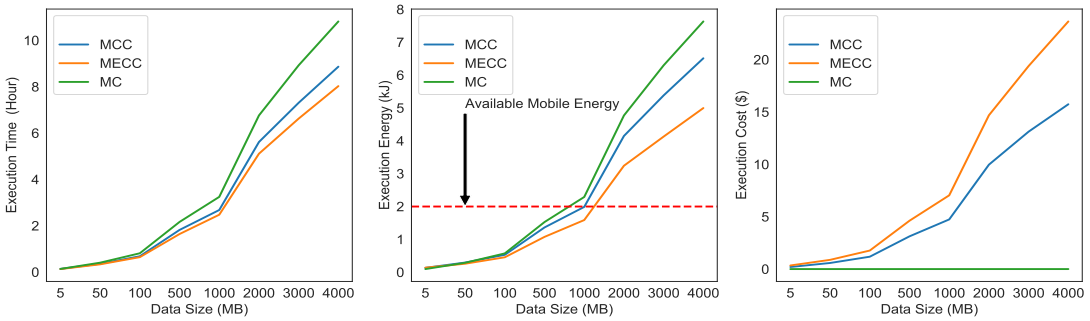
low-bandwidth network is used. For MC, the processing energy is the only factor considered, because no data transfer is required. For MECC and MCC, the low bandwidth increases the data transfer time, and hence the data transfer energy. On the other hand, with the 4G network, in Figure 5.11, the high bandwidth saves energy by sending some IoT application tasks to nearby edge nodes and reducing the energy consumption overhead of the local execution. In addition, as shown in Figure 5.12, the availability of a WiFi connection is a huge incentive to run an IoT application locally in a user's device because the WiFi connection potentially reduces in data transfer energy and cost when data is collected from sensors. Data collection with an edge node is discussed next.

Figures 5.13, 5.14 and 5.15 show the experimental results of an IoT application execution when IoT data is collected at the edge layer. The high performance of running the application in an MECC environment was expected. The optimiser was able to find an optimised solution by running the workflow at the edge layer. This scenario is promising when the mobile network is unstable or unpredictable. In addition, Figures 5.14 and 5.15 show that the application can be run in MECC and MCC with large data files without violating the energy constraint.



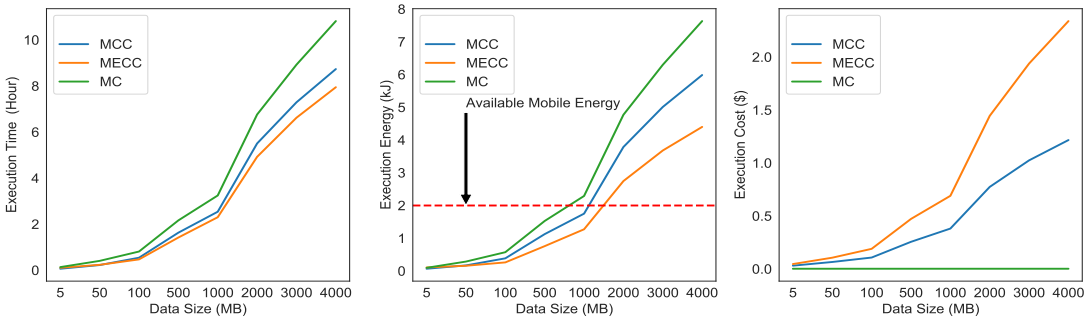
(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.10: IoT application with mobile data collection : 3G network



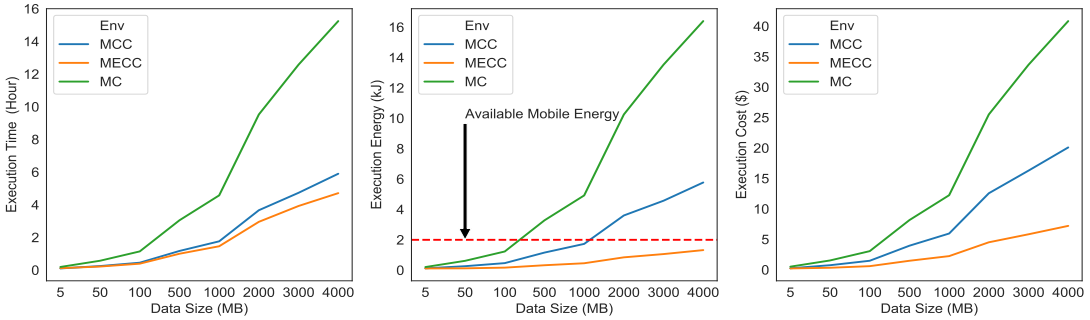
(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.11: IoT application with mobile data collection : 4G network



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.12: IoT application with mobile data collection : WiFi network

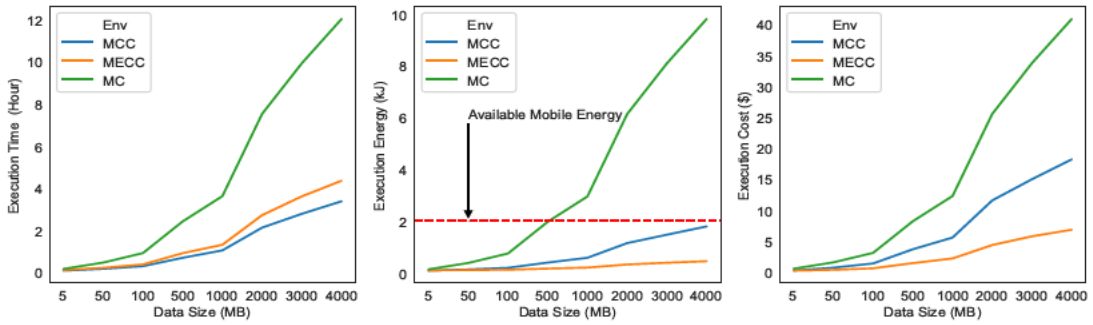


(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.13: IoT application with edge data collection : 3G network

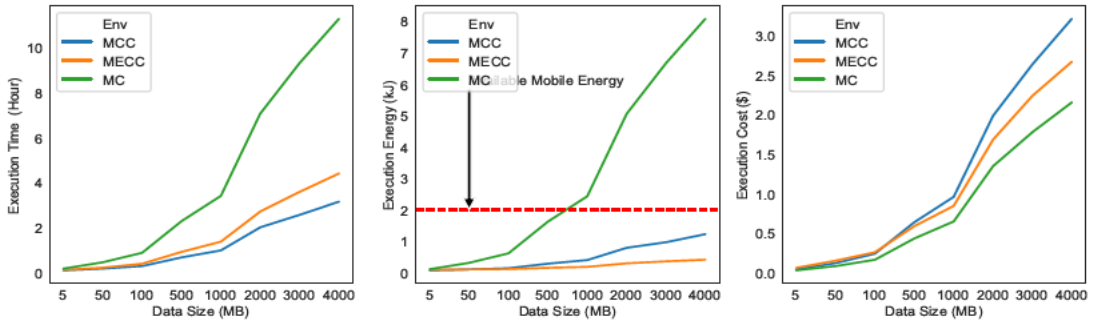
5.5 Discussion and Recommendations

This section presents the main insights from the experimental results outlined in Sections 5.3 and 5.4. The results demonstrate the contributions of the studied parameters, namely, application type, network quality and data size, to the offloading optimisation decision. The main insights are as follows.



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.14: IoT application with edge data collection : 4G network



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 5.15: IoT application with edge data collection : WiFi network

1. Selection of a computing environment to reduce energy consumption and monetary cost is highly dependent on the amount of data to be transferred over the communication network. This factor is critical for applications that require heavy data communication with distributed data storage. For example, in BoT applications, edge resources can be effective, particularly with low network quality.

2. There is promising potential for use of edge resources with on-edge data collection. IoT applications involve collecting data from a large number of sensors. For low capability MC, the process of collecting data and acting as a data source is not energy efficient, and costly with cellular data usage. Thus, the availability of edge nodes which can communicate with IoT sensors for data collection allows offloading optimisation to reduce data transfer time and cost.

3. The data dependency between application tasks plays a significant role in resources allocation planning. For example, in workflow applications, the increase in data communication overhead to transfer large data files motivates the optimiser to adopt an in-place allocation strategy. This means moving the computation close to the data location. This strategy demonstrates viable optimisation results because the allocation targets the closest and highest-capability computation resources.

Based on these insights, the following recommendations for execution of data-intensive mobile applications can be made.

- Data-intensive mobile applications should be handled using MECC. This is the most efficient architecture, particularly for loosely coupled applications, such as BoT, in which data and computation dependencies are low. MECC allows an optimised computation distribution over edge and cloud resources. However, MCC also reduces BoT execution time substantially with high bandwidth networks to offload large data and complex task to powerful cloud resources.

- For workflow applications, MECC provides valuable capability for energy-sensitive scenarios. Results show that as data transfer between workflow tasks increases, the optimal strategy is to perform computation with cloud-only or edge-only resources. The availability and quality of the mobile network are the determinants of strategy selection.

- For IoT applications, the ability of users' devices to collect data from IoT sensors is critical for selecting the best computation environment. With low bandwidth networks, local processing is the preferred option when data is stored in local devices.

- The adoption of an external mobile energy source would be an effective enabler of running data-intensive applications on user devices. Resolving the energy shortcoming can strength the opportunities of mobile-based and edge-based computation systems.

- The performance analysis highlights that high proportion of running data-intensive application cost is related to data migration. Thus, more sophisticated file management and data replication schemes can be applied for efficient usage of storage capabilities at the edge layer.

5.6 Summary

Rapid and profound advances in the mobile telecommunications industry have created the need for mobile-aware computing models that can benefit from the capabilities of resources at different computation layers such as the cloud, fog and edge. Moreover, integration between computing models supports the broad engagement of mobile devices with various application model, such as BoT, workflow and IoT.

This chapter describes the variation in computing and application models and how this can impact the offloading decision, that is, how application tasks can be allocated to resources. For data-intensive mobile applications, the work included a set of parameters to determine optimisation decision viability on each computing model, including application data size and maximum execution time, network quality and device energy. The results demonstrate the ability to optimise data-intensive application execution while considering the target computing environment and problem constraints. Major insights are: MECC is the preferred target for loosely coupled applications or BoTs; network quality is a critical factor in the decision process e.g. (with WiFi connection, MCC is the optimised target for workflow applications); for large data size inputs, adopting edge computing supports nearby computation to reduce data transfer overhead in terms of network latency, energy and cost.

For the purpose of offloading optimisation, an offloading framework was developed, that allows interaction between a set of components for handling services like offloading optimisation, resource communication and task execution. The next

chapter discusses the computation offloading framework and all its components and services.

Chapter 6

Implementation and Simulation Environment

Chapter 3 provides an offloading optimisation technique for a hybrid MCC system by leveraging three resource layers of mobile devices, cloudlets and the cloud. In Chapter 4, the offloading system was improved by incorporating the edge layer to revolve issues of network latency and computation performance closer to the user layer. In Chapter 5 the selection of a computing system to obtain the minimum optimisation value is described. Several parameters are involved, including data size, network bandwidth, cost, and the type of application. This chapter provides a framework for profiling and simulating the running of data-intensive mobile applications on mobile-aware computing systems. The framework consists of a set of components which perform profiling, resource investigation and communication, context monitoring, offloading decision-making, and QoS optimisation.

6.1 Introduction

In the last decade, the mobile industry has gained huge attention to empower mobile devices to engage in daily business and social activities to benefit from non-stationary, connectivity and accessibility features in many disciplines, like social

communication, E-commerce, smart applications, scientific experimentation, etc. This widespread involvement steers the growth of mobile computing industry to achieve the requirements of new application models. As noted in previous chapters, the number of MC users is increasing and the volume of data generated from user's devices is increasing exponentially. These challenges necessitate the development of adaptive resource and computing frameworks to overcome the deficiencies of mobile devices related to energy limitation, storage capacity and computation capability. The fundamental concept in overcoming challenges of mobile devices is sending computation-intensive tasks for remote processing [24, 25]. Early research on offloading optimisation was focused on energy reduction, with many techniques adopted, such as code offloading [119] and VM migration [120].

This thesis proposes techniques and algorithms for the execution of mobile applications in mobile-aware computing environments. Allocating application tasks to computation nodes (local, edge, or cloud) is formulated as an optimisation problem, the solution of which is the minimum energy consumption and cost under the constraints of available energy and task deadlines. The first optimisation objective is to reduce the energy consumed by the mobile device. Energy consumption was studied from three perspectives: energy for processing task(s), energy for transferring and receiving data from other computation units and storage, and holding (device waiting) energy. The second optimisation objective is monetary cost. To implement the proposed offloading algorithms and related resource management services, the candidate developed an offloading framework.

This chapter explains the optimisation framework, covering the main components and communication between components, data collection, profiling, cost and time estimation, optimisation, application execution services and optimisation steps. The rest of the chapter is structured as follows. Section 6.2 describes the high-level system architecture and the offloading framework. The system implementation is highlighted in Section 6.3 and the offloading framework validation is presented in Section 6.4. Finally, Section 6.5 summarises main chapter findings.

6.1.1 Designing a Data-Intensive Applications Offloading System

The application offloading system represents the set of activities required to perform application offloading planning on a given computing system. Figure 6.1 shows a high-level system use case model. The system has two main actors. After downloading the offloading management app, the user should register to the system. Next, the user needs to select preferred application templates, which reflect the execution of application types of BoT, workflow and IoT. The user starts the offloading planning process by adding an application, configuring it to set QoS requirements, and finally setting up input data sources. The admin is responsible for approving received applications based on user status, input resources validation and application structure format. After validating the app, the offloading planning process start. The offloading plan is then sent for execution. The admin is also responsible for monitoring app execution.

As seen in the previous chapters, data size has a non-trivial impact on offloading

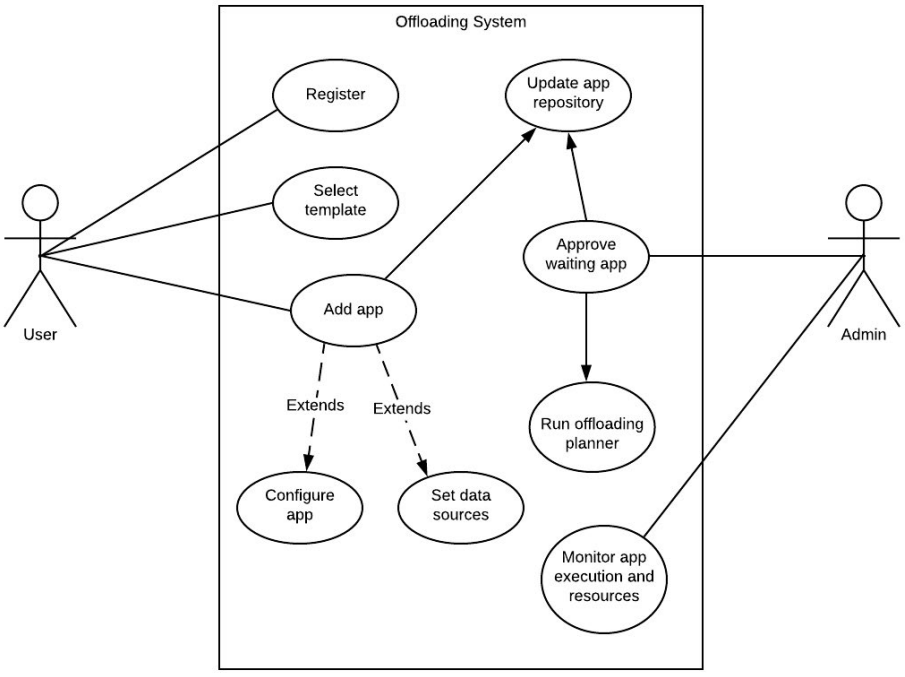


FIGURE 6.1: High-level offloading system

decisions and raises questions about how to reduce the implications of large data size for device energy, monetary cost and optimisation model complexity. The following points summarise the core requirements of designing a data-intensive offloading system which is able to overcome the aforementioned issues.

1. *QoS-aware.* The offloading system should include a QoS-aware technique to support joint optimisation of objectives like device energy, monetary cost, task deadline and task waiting time.

2. *Collaborative.* The offloading system should implement a resource allocation strategy to benefit from the advantages of heterogeneous computing models, including mobile, edge and cloud.

3. *Adaptable.* The offloading system should be adaptable to changes in context and computing environment parameters. Such parameters to consider are the mobile network and bandwidth, number of users, resource availability, application model, data size and user QoS constraints.

6.2 High-Level Offloading System

A data-intensive mobile application offloading system is a set of services which allows a user to submit and run a mobile application in an MECC environment. Figure 6.2 shows the high-level offloading system. System services are described as follows.

- *Submitting an application.*

  The user submits/sends an application to the offloading system. The application should have a certain format to enable extraction of application details for decision-making and execution. The application handler is responsible for receiving and analyzing the application to extract required data about application tasks, which includes data input and output locations, pre-tasks and post-tasks, computation complexity and execution deadline. Next, the application is saved in a local database.

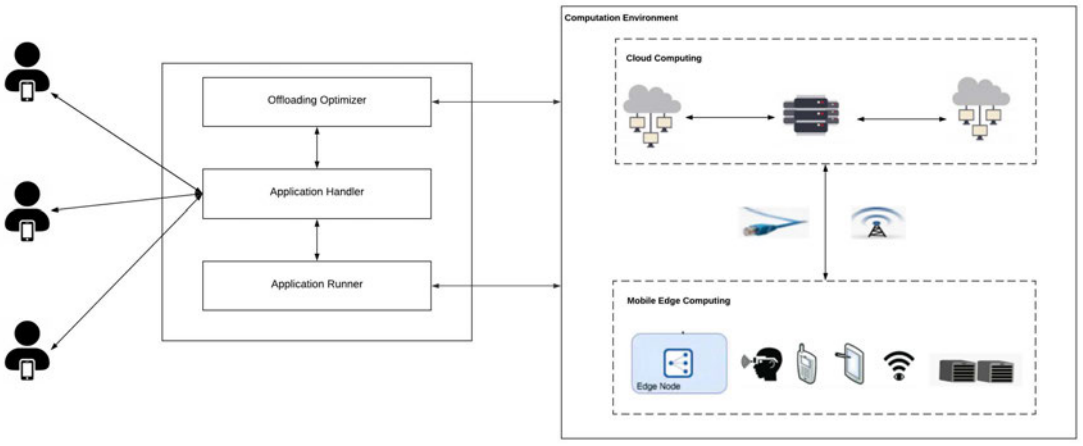


FIGURE 6.2: High-Level Offloading System

- *Offloading Optimiser.*

  The optimisation service is provided by the offloading optimisation framework, which is a set of components, each with a specific function. Figure 6.3 illustrates the framework structure and the interactions between its components.

  - *Application Profiler.*

    The offloading decision to allocate application tasks to resources is subject to the expected energy consumption of the user's device and the estimated cost of running the application. Because the decision-maker handles a variety of tasks which differ in computation complexity and programming logic, profiling is essential to recognise how task execution affects energy and cost consumption. The profiling process aims to measure the sensitivity of task execution to parameters of data size and network bandwidth. In other words, it is designed to determine the impact of change in input data size or network bandwidth on energy and cost consumption. Energy profiling relates to the functions of task computation at the user's device, data transfer between the user's device and the external data source, and waiting for remote execution. The task t profiling process is described in Algorithm 5. The processes represent running the task t on resource r_{targt} with input data d_{input}.

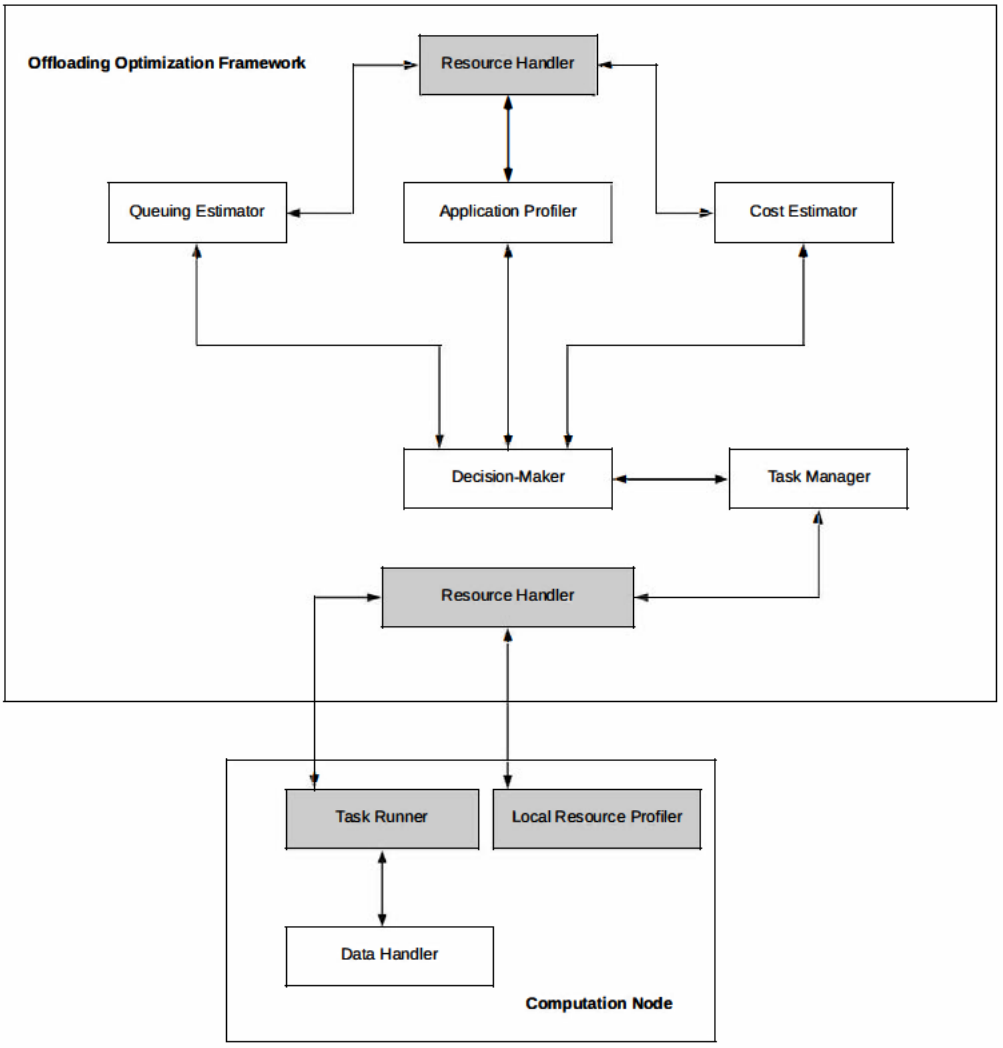


FIGURE 6.3: Offloading optimisation framework

To obtain acceptable profiling accuracy, the profiling processes are repeated x times for each task. At each iteration, random resource r_{target} is selected from the list of resources R, Line 7. Three types of resources are available, namely, local device r^l and an edge node r_i^e, a cloud server r^c. Next, a random file is selected from dummy files list D, and the task sends for execution, Lines 8 and 9, respectively. Finally, the execution result is collected and the task profile record is updated, Line 10. In addition to this result, context data is recorded, including details of the

Algorithm 5 Task profiling pseudocode

1: **Inputs:**
2: Application tasks $T = \{t_i, ..., t_n\}$
3: Computation Resources $R = \{r^l, (r^e_1, ..., r^e_m),\ r^c_1\}$
4: Data files $D = \{d_i, ..., d_n\}$
5: **for** t in T **do**
6: **for** i in {0..30} **do**
7: $r_{target} = selectTargetResource(R)$
8: $d_{input} = selectInputFile(D)$
9: $result = runTask(t, r_{target}, file)$
10: $updateTaskProfileRecord(t, result)$
11: **end for**
12: **end for**

data communication network and bandwidth, and device energy and processing capacity. The application profiler allows access to profiling data, and sends tasks for execution via service API.

– *Resource Handler.*

The resource handler is the external gateway that allows the optimisation framework/engine to communicate with computation and storage resources. It has a database to store details about these resources, including computation capacity, available storage and access APIs. Before task profiling or application execution, the resource handler collects data about user device energy, connected edge resources, and network and bandwidth status between user device and external resources.

The resource handler was implemented to include two types of service APIs: resource APIs are responsible for fetching and sending status requests to resources; data access APIs allow other components to collect sufficient details about resources for cost and task waiting time estimations.

– *Queuing Estimator.*

It is assumed that many users can submit their application to the framework for execution, and share computation resources. Thus, with the existence of deadline constraints on application tasks, the decision-maker should be aware of the task waiting time for processing in edge

and cloud servers. To control task waiting time, a $G/G/1$ queuing system was implemented, in which task arrival rate λ and task processing μ follow a general distribution.

- *Cost Estimator.*

  Optimisation decisions are designed to solve a joint minimisation problem: how to reduce the total energy consumption and cost. According to the optimisation problem formulation, both optimisation variables depend on time calculations, which include time for local processing, remote processing, waiting after offloading all tasks, and transferring data to and from the user's device. Moreover, the cost estimator communicates with the resource handler to get information about available resources, as well as context information related to the mobile network and bandwidth. Next, energy and cost calculations are applied for each task, and returned to the decision-maker.

- *Decision-Maker.* The decision-maker is the core component of the optimisation engine. To make a decision, the decision-maker uses an offloading optimisation technique, such as PSO or MLIP, to search the problem space and generate offloading plans (paths) and then select the optimal one, based on the optimisation constraints of energy and deadline. The decision-maker runs the optimisation process based on resources data from the resource handler and estimation values from the cost and queuing estimators.

- *Task Manager.* The task manager is responsible for managing the execution of application tasks. It receives processing requests from the decision-maker for application profiling and execution purposes. The task manager records task processing status, which helps the application profiler to update profiling measurements for energy and data sensitivity. To run a task, the task manager sends a request to the application handler, which in turn sends it to the application runner.

- *Application Executor.*

  The application performs the real communication with computation and

storage. A set of of task execution APIs to run application tasks based on task manager requests were implemented.

6.3 System Implementation and Deployment

To meet the requirements of the offloading system in the edge cloud computing deployment architecture, the processing framework was implemented at the user's device and a remote server. Figure 6.4 shows a high-level implementation architecture for the proposed offloading system. The system implementation is provided

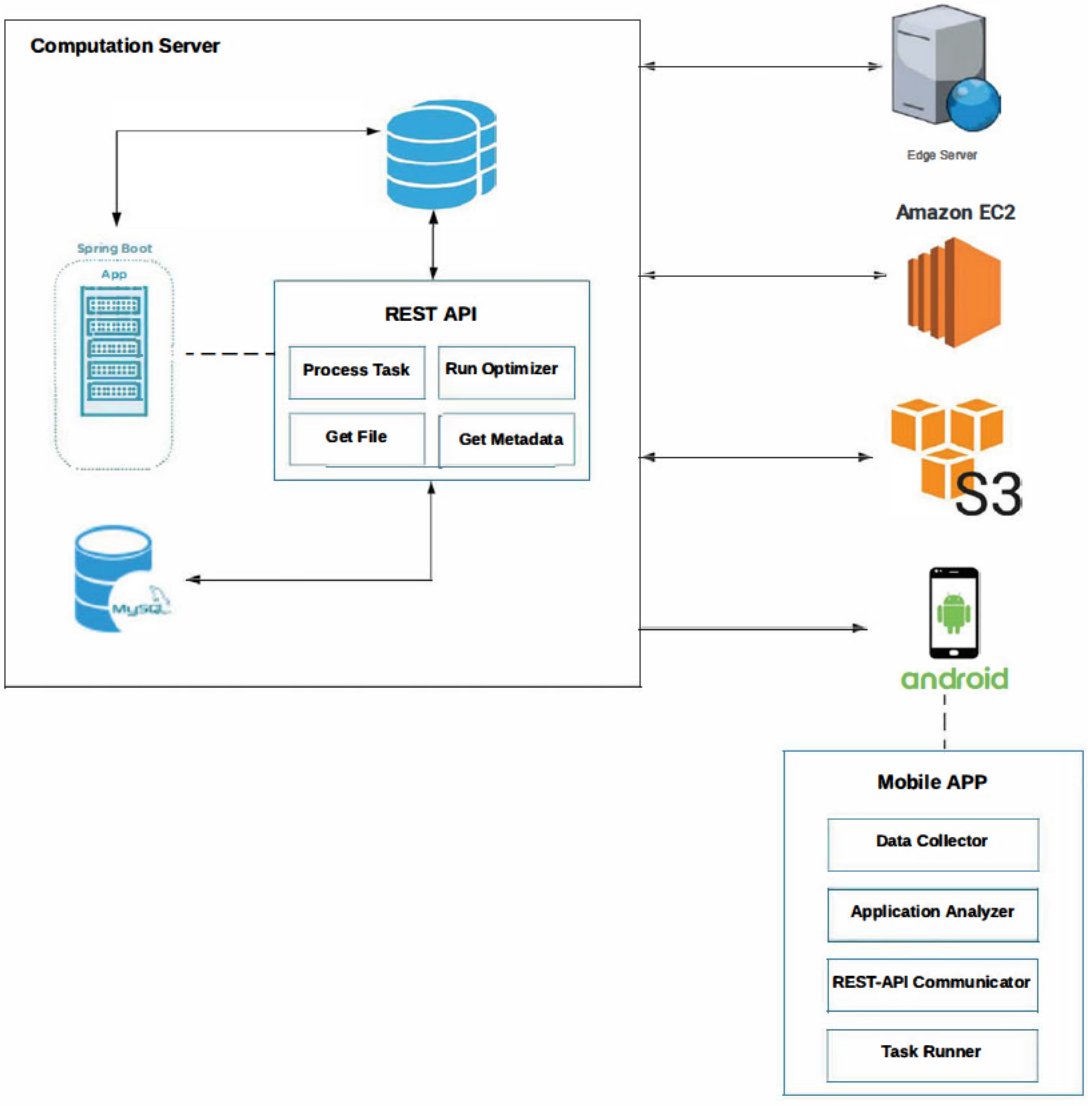


FIGURE 6.4: Offloading system implementation architecture

through two types of applications: a mobile application (at the user's device), and a server application developed by Spring Boot Java Framework.

- The mobile application.

  An Android application was developed to provide the following services in the context of the offloading system.

  1. Data collection.

     The mobile application can use the sensing capability of the hosted device to communicate with and collect data from sensors and IoT devices. However, this feature is implemented only partially for the sake of applying the model.

  2. Application analysis.

     Application specification is formatted in a JavaScript object notation (JSON) file to facilitate setting task configuration and dependencies. A JSON formatter and Validator was developed to read the application file and convert it to a system-readable object.

  3. REST-API communication.

     The mobile application needs to communicate with external entities for many purposes, including sending application specifications for offloading to the optimisation server, requesting/sending data files, and sending/receiving task processing results.

  4. Task processing.

     As a part of the MECC environment, a mobile device is considered a computation unit for task processing. Thus, the author implemented a task runner module to run application tasks using Java Multithreading feature.

  The mobile app represents the user's interface with the system. The app was developed in Android Studio. Moreover, a local SQLite database was created for profiling purposes.

- REST-API Server Application.

  At each remote server, a Spring Boot framework was installed in all access-to-server services over HTTP protocol via RESTful web services. The Spring Boot framework provides easy development and a scalable solution [121].

  1. Run optimiser.

     Run optimiser is the service API and interface for the optimisation engine illustrated in Figure 6.4. Optimisation engine components are basically back-end functions managed by the optimisation manager. The optimisation API receives an application configuration and returns the offloading decision, which reduces energy consumption and cost.

  2. Process task.

     The process task API allows the system to receive tasks for processing from other computation units based on the offloading plan. However, in some cases, such as workflows and IoT applications, a server can receive a BoT to be processed sequentially or concurrently. At the server back end, we implemented a task processing framework which contains functions to validate and run the task, get data files, exchange data between tasks, and update the optimiser with execution results.

  3. Get a file.

     The proposed offloading system assumes that task data files can be stored in various locations. 1) Cloud storage: files can be located in cloud VMs (EC2 machines) or in S3 (the object storage service from AWS). 2) Edge storage: it is assumed that data files can be stored in edge servers temporarily for the purpose of saving data transfer time and cost for cloud communications. 3) Local storage: a user's device can provide data storage, particularly when tasked with sensing and collecting data from sensors and IoT devices.

  4. Get metadata.

     As mentioned in the previous section, the resource handler is responsible for collecting data about environmental resources. The resource handler sends a request to get resource details and calls the get metadata API.

For storing data profiles and user profiles, the application server offers data APIs to communicate with a database server which runs the MySQL database.

6.4 Comparison of The Proposed Framework and Real Framework Results

In order to validate the proposed framework, we have compared framework results with real-time application execution. In this thesis a data-aware offloading framework was provided to facilitate the planning of application execution, which has high and active data migration within application tasks. One challenge related to data-intensive application offloading is the unpredictable behaviour of mobile networks. Chapters 3 and 4 provides modelling for estimating optimisation objectives of execution time, energy consumption and monetary cost. To validate the proposed framework, we have developed a profiling framework to support the offloading technique in estimating the optimisation objectives for varying input data and network bandwidth. To the best of our knowledge, this research is unique in providing an approximation for offloading optimisation sensitivity to the given application model. Figure 6.5 shows the standard approach for simulation model verification and validation.

6.4.1 Conceptual Model

Conceptual modelling is the abstraction of a simulation model from a real-world system. When modelling an application, it is important to decide which part(s) and function(s) are to be simulated. However, modelling all program parts is an extremely complicated process. To model the application execution behaviour, we have identified three parameters, namely number of processing instructions in MIPS, CPU load, and data execution sensitivity factor. The "number of instructions" parameter refers to the processing load for a task in a single-core machine, and the CPU load refers to the percentage of CPU cycles used for running the

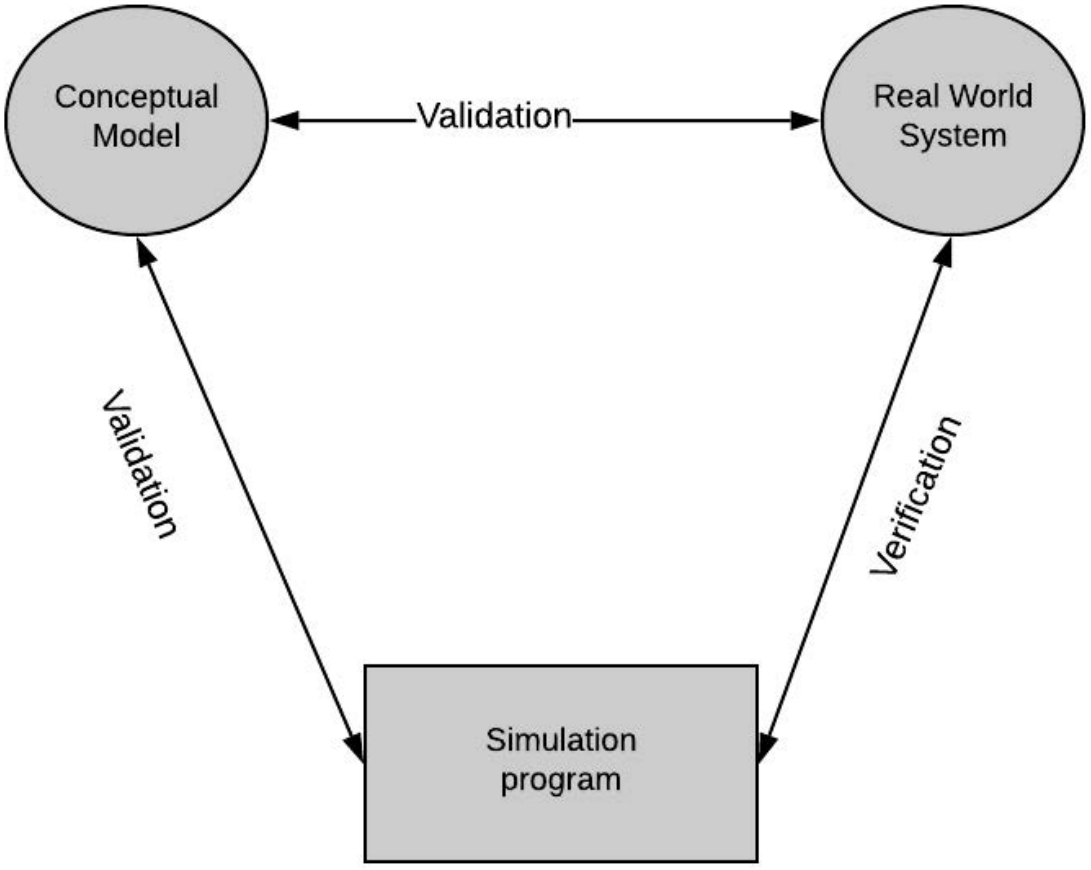


FIGURE 6.5: Model verification and validation

task. The data sensitivity factor determines the impact of data size increase on task processing load. With this modelling, different types of execution models could be constructed. The computation-intensive model embraces high number of MIPS and high CPU usage, where as the data-intensive model corresponds to large input data size as well as high processing sensitivity to increased data size. Table 6.1 shows an example of modelling a 5-tasks application.

The calculation of task processing time PT is formulated in Eq. (6.1).

$$PT = (\frac{I}{w} + (s.\omega))\zeta \tag{6.1}$$

TABLE 6.1: Mobile application modelling

ID	Instructions (I)	CPU load (ζ)	Data size in (MB) (s)	Data sensitivity (ω)
1	212	0.6	390	0.134
2	160	0.4	150	0.165
3	240	0.9	936	0.437
4	190	0.1	613	0.122
5	100	0.8	233	0.332

6.4.2 Model Validation

This section describes the outcomes of validating the proposed data-intensive offloading framework in respect to real-time application execution. For the purpose of model validation and application profiling, the experiment setup provided in section 3.5.1 is applied.

- *Data sensitivity profiling*

  The data sensitivity factor ω determines how task processing load increases with increasing input data size. This concept is critical for data-intensive applications which handle large data files for either processing or migration. To get an accurate measurement of the sensitivity factor, the correlation between task input data size and the change in processing time was analysed. We conducted 30 experiments to establish accurate estimation of task complexity and the sensitivity factor. Figure 6.6 shows the data sensitivity factor values for the application tasks in the sample model in Table 6.1.This showed that the sensitivity value is only correlated with the task processing logic.

- *Energy profiling*

  Energy profiling aims to examine the effect of change in data size on both data communication and processing energy. The energy profiling includes two parts. The former is processing energy, while the latter is data communication energy. PowerTutor software was used for power estimation, giving energy consumption estimates within 5% of actual values [105]. Data communication energy profiling is critical due to the existence of different network interfaces. To assess energy profiles we performed an experiments

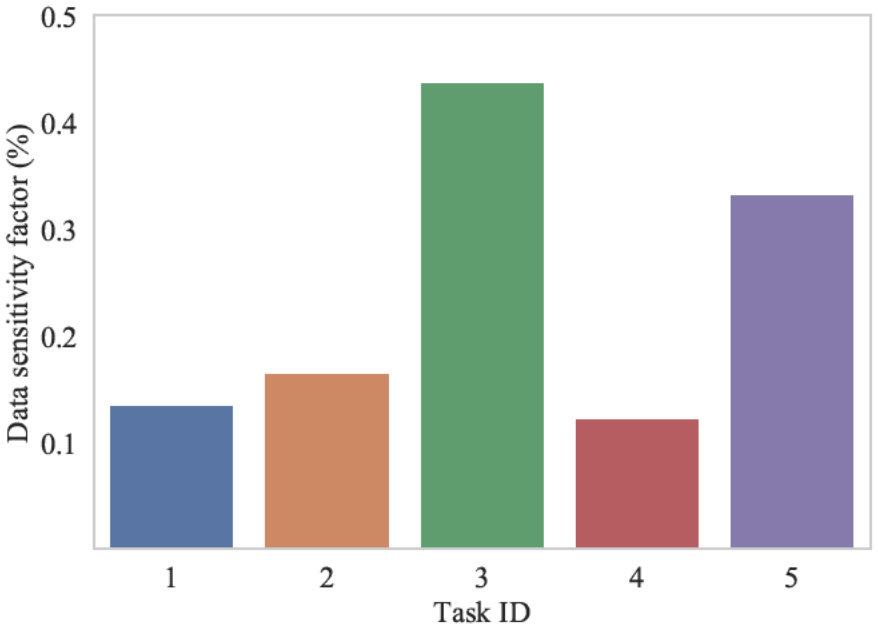


FIGURE 6.6: Processing energy profiling

on three mobile network interfaces, WiFi, 3G and 4G. Figure 6.7 shows the results, in terms of energy consumption, of transferring file of a 100 MB data size over the three network interfaces. Overall, WiFi connection consumes significantly less battery power than 4G or 3G cellular networks. In addition, WiFi connection demonstrates less profiling variation due to the expectation of highly stable connection in a closed area coverage. In contrast, with 4G, high variance in profiling results is apparent under different networking conditions and coverage areas.

In addition we studied data communication energy consumption for each network interface over time. This profiling enables accurate offloading decisions over long application execution time. Figure 6.8 illustrates that WiFi connection is highly stable and less sensitive to change over the connection time than 3G or 4G. On the other hand, 4G and 3G connection stability can be affected with minimum of 37% and maximum of 48% of connection quality. Lastly, Table 6.2 presents the 95% confidence intervals (CIs) for data communication energy profiling. Moreover, the table shows the number of iterations needed to achieve the confidence level. WiFi communication required fewer iterations due to its high stability compared to 4G and 3G.

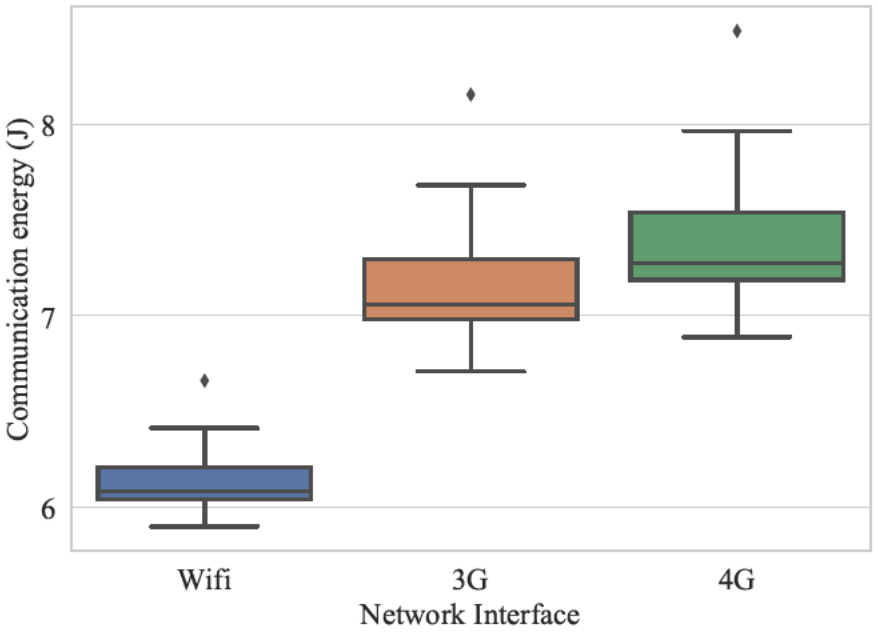


FIGURE 6.7: Data communication energy profiling for 100MB file size

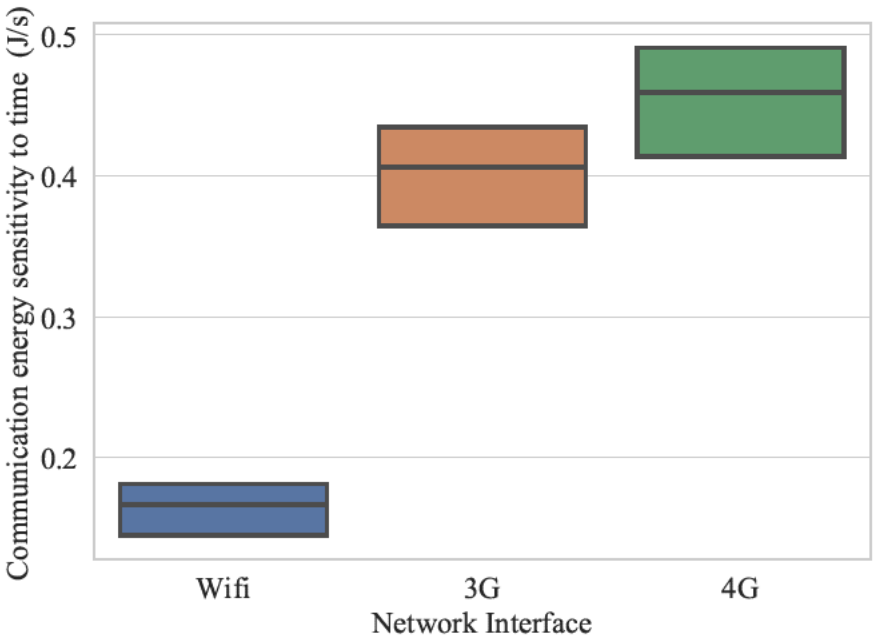


FIGURE 6.8: Data communication energy sensitivity profiling for 100MB file size

TABLE 6.2: 95% CIs for data communication energy profiling

Network Interface	Lower and upper 95% CI (J)	Number of iterations
WiFi	10.35 - 12.45	18
3G	19.67 - 23.65	30
4G	21.74 - 26.14	30

For processing energy, the main objective is to study how task complexity is related to the energy consumption of the mobile device. Three complexity levels -low, medium and high- were studied. The complexity level refers to both CPU usage and input data size. For example, in Table 6.1, tasks 5, 2 and 3 are examples of low, medium and high complexity tasks, respectively. Figure 6.9 highlights the differences in processing energy profiling for the three levels of complexity. Firstly, the results demonstrate the expected steady increase in amount of energy consumed as complexity increases. Secondly, the profiling process with 30 experiments for each complexity level, shows an acceptable level of stability with low variance and no significant outliers.

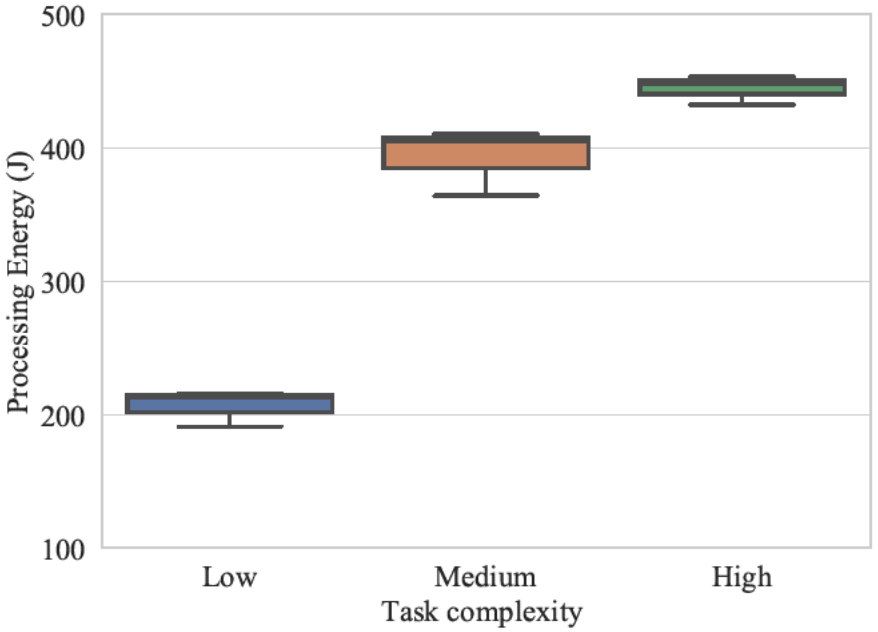


FIGURE 6.9: Processing energy profiling

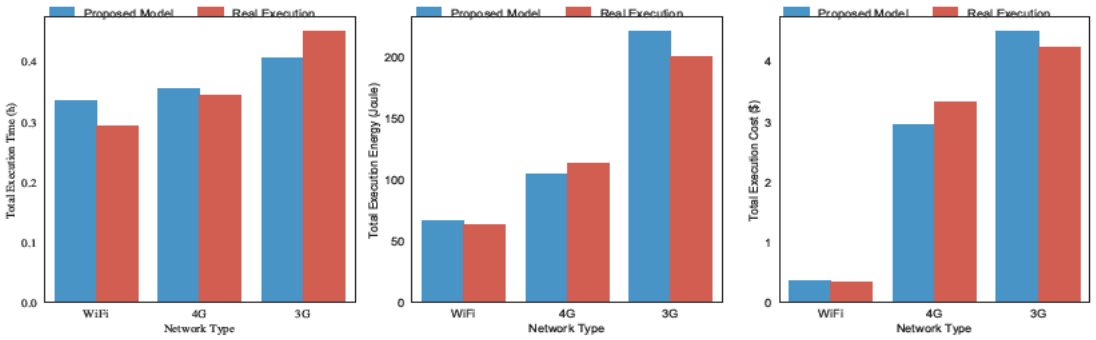
- *Framework validation*

  This section provides the validation for the proposed execution model using a real-time execution scenario. Validation was based on running an application with 30-task of small data model using both the proposed model and real implementation for three communication networks. The validation was applied for the two computation frameworks hybrid MCC and MECC, as shown in Figure 6.10 and Figure 6.11, respectively.

For validating the proposed model, offloading optimisation results were compared with those obtained from a real execution scenario involving three communication networks (3G, 4G and WiFi) and three performance metrics: execution time, mobile energy and monetary cost. For the purpose of validation, only the small data scenario was implemented. We implemented a profiling process to compute an approximation of energy and bandwidth estimation parameters. Prior to a profiling iteration, an experiment configuration is obtained from data size distribution. For each configuration setup, a BoT application of 30 tasks is executed, and averages of processing, communication and waiting energy values are recorded for 30 executions.

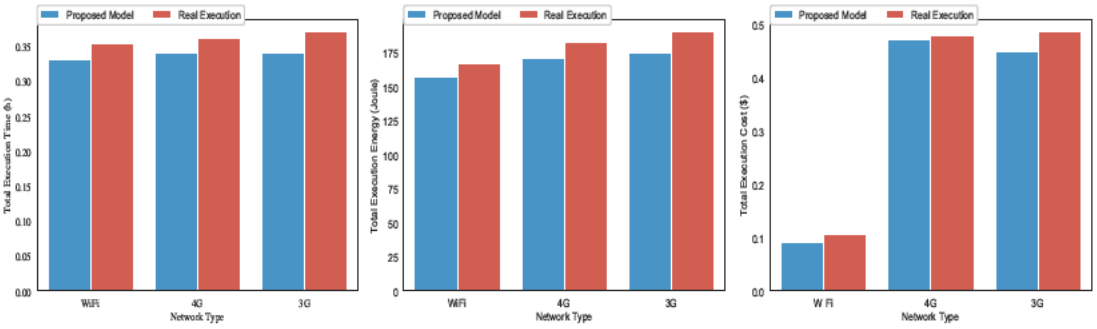
Figure 6.10 shows results for validating the proposed offloading framework in the hybrid MCC environment. The results include some discrepancies between the proposed model and real execution scenario due to the differences in available bandwidth and network latency. For execution time, as shown in Figure 6.10 (a), the proposed model scenario demonstrates higher values than the real-time scenario with high-bandwidth networks. This can be explained by the high variation in bandwidth capacity, which occurred because the experiment was performed in different geographical areas. Overall, the highest estimation error was observed in the 3G network. The average errors for the execution model are 8% for execution time, 11% for energy consumption and 15% for monetary cost. Based on average estimation errors, it can be concluded the that proposed execution model is accurate enough to use for offloading in the hybrid MCC system.

Figure 6.11 shows the results obtained from experiments performed to validate the proposed offloading framework in the MECC environment. The evaluation results for the three model parameters are based on the change in the mobile network interface. The experiments predict slight estimation errors for the three model parameters, with 7%, 7% and 12% for execution time, energy consumption and monetary cost, respectively. On average, the estimation error per scenario is around 9%.



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 6.10: Evaluation of proposed optimiser with real scenarios for three different communication networks: hybrid MCC case



(a) Execution time (b) Mobile energy (c) Monetary cost

FIGURE 6.11: Evaluation of proposed optimiser with real scenarios for three different communication networks: MECC case

As in the hybrid MCC case, the MECC results include some discrepancies between the proposed model and real execution scenario due to differences in available bandwidth and network latency. For execution time, as shown in Figure 6.11 (a), the real-time scenario demonstrates higher values than the proposed model with high-bandwidth networks. This can be explained by the high variation in bandwidth capacity, once again because the experiment was performed in different geographical areas. Overall, the highest estimation error was observed in the 3G network.

6.5 Summary

Data-intensive mobile applications bring significant challenges for offloading optimisation related to processing large amounts of data and transferring large datasets over cellular and mobile networks. This chapter provides a detailed description of the proposed offloading system, which has the required interfaces and components to run the offloading process and communicates with external resources for the purposes of model parameters estimation and profiling. Moreover, the chapter provides some technical information on implementing front-end and back-end applications. For framework validation, the chapter provides comprehensive discussion on application conceptualisation, energy profiling and model validation against real model execution. Processing and data communication energy profiling results show low estimation variance. Finally, offloading model validation in both MCC and MECC environments demonstrates low estimation error, averaging less than 10% among all optimisation parameters.

Chapter 7

Summaries and Discussion

This chapter summarises the key findings of this thesis in relation to data-intensive mobile application offloading in different computing systems. In addition, the chapter highlights future directions for research into data-aware offloading techniques and frameworks.

7.1 Discussion

Chapter 2 provided an extensive survey of the literature on many aspects of computation offloading and device augmentation. A large proportion of the research in this field to date is about developing energy-based solutions by integrating cloud services and extending the capability of user devices through hybrid and collaborative resource models such as MCC, MEC and MECC. Chapter 2 showed that considerable research effort has been directed towards computation-intensive mobile application offloading optimisation, but little attention towards data-intensive offloading optimisation. However, as noted earlier, emerging data-intensive applications, such as face recognition and natural language processing, impose challenges on MCC platforms because of high bandwidth cost and data location issues.

The literature analysis concluded that research to date has gone in one of two directions: application complexity and structure variation, or the contribution of data

size parameters to offloading optimisation decision-making. Previous researchers strove to overcome the challenges of data-aware mobile application offloading. However, their models do not consider the contribution of data size variation in association with other optimisation parameters in application offloading decisions. In addition, these models do little to integrate cloud and edge systems to resolve data-intensive application offloading and scheduling optimisation. Moreover, although the literature presents comprehensive research proposing computation-intensive and data-intensive application offloading optimisation techniques from different perspectives, on a variety of computation environments, these issues are not fully addressed.

Chapter 3 proposed a data-intensive mobile application offloading optimisation framework for a hybrid MCC environment. The hybrid computing model consists of the public cloud, cloudlets and mobile devices. This integration can meet data-intensive offloading requirements by offering powerful resources at the cloud layer and in highly accessible and distributed cloudlets. These resources can be accessed via WiFi or the cellular network. Moreover, the chapter outlined a QoS-aware resource allocation model for scheduling application tasks in a hybrid MCC environment. The model aims to generate an allocation plan that provides the best optimisation for device energy usage and monetary cost of remote execution and data communication. To find the best allocation plan, PSO [101] was adopted to search the offloading plans space by an iterative improvement to candidate solution until finding the global solution.

Results demonstrate the technique's ability to generate adaptive resource allocation in response to variation in application data size and network bandwidth. The proposed technique improves the execution time for data-intensive applications by an average of 78%, and reduces mobile energy consumption by an average of 87% compared to a mobile device alone, while monetary cost increased only 11% due to using cloud resources and mobile communication.

Emerging data-intensive applications pose challenges for MCC platforms because

of high latency, cost and data location issues. Chapter 4 outlined a study of adopting edge computing through a hybrid model of MECC. In addition, One issue with evolutionary optimisation techniques like PSO is the exponential increase in search complexity, particularly with large-scale applications. The chapter investigated use of an offloading optimisation with linear search technique, MILP, to reduce the search time complexity of the PSO technique. The optimisation model is formulated as a MILP problem, which considers both monetary cost and device energy as optimisation objectives. Moreover, the allocation process considers parameters related to data size and location, data communication costs, context information and network status. To evaluate the performance of the proposed offloading algorithm, the chapter present real experiments on the implemented system with a variety of scenarios, such as different deadline and multi-user parameters. The proposed offloading technique is compared to PSO and local execution techniques.

Results showed high model ability to respond to the high fluctuations in application data size and network bandwidth it reduced the execution cost of data-intensive applications by an average of 46% and 76% in comparison with PSO and full execution on a mobile device only, respectively. In addition, the new technique reduced mobile energy consumption by 35% and 84%, compared to PSO and full execution on a mobile device only, respectively.

Chapter 3 and 4 proposed algorithms and techniques for data-intensive application offloading on hybrid MCC and MECC systems. The experimental work presented in these chapters focused on a single type of application-BoT. Even though BoT applications are convenient to validate and evaluate the proposed techniques, but other application features like structure and data dependency are need to be elaborated. In Chapter 5, we considered these features and provided a comprehensive analysis of many aspects of mobile-related computation models. These features determine the dependency between application tasks in terms of computation and sharing data. Application complexity and structure should be considered while planning for offloading to a mobile-aware computing environment. In combination with the size of the application data, and the quality of the communication

network, the application structure adds another dimension to the offloading optimisation process.

Chapter 5 provided recommendations for selecting a mobile-aware computation paradigm to execute data-intensive mobile applications based on aspects such as application model, data size, mobile network performance, and mobile energy status. The selection of a computation model is based on its energy consumption and the total monetary cost of executing application tasks and transferring data between computation nodes. Hence, the chapter provides a comprehensive analysis of how to select the best mobile-aware computing system to offload a data-intensive application based on parameters of data size, network quality and application structure. The results demonstrate that it is possible to optimise data-intensive application execution while considering the target computing environment and problem constraints.

The main insights from the results presented in Chapter 5 are: (1) MECC is the preferred computing system for loosely coupled applications like BoTs, where the network performance makes a critical contribution to reducing the cost overhead of offloading large data files; (2) for workflows, MCC has the best performance because most processing can be undertaken at the cloud layer; and (3) there is promising potential for use of edge resources with on-edge data collection and communication with a large number of IoT devices. For low-capability mobile devices, the process of collecting data and acting as a data source is not energy efficient, and costly due to cellular data usage. Thus, the availability of edge nodes which can communicate with IoT sensors for data collection allows offloading optimisation to reduce data transfer time and cost.

For the purpose of offloading optimisation, an offloading framework is proposed to allow interaction between a set of components for handling services like offloading optimisation, resource communication and task execution. Chapter 6 provided a framework for profiling and simulating the execution of data-intensive mobile applications on mobile-aware computing systems. The framework consists of a set

of components which perform profiling, resource investigation and communication, context monitoring, offloading decision-making, and QoS optimisation. The chapter also discuses the high-level offloading system that represents end-to-end service interactions between end users, the core optimisation frameworks and the computation system.

In addition, the chapter presents technical details on implementing and deploying the proposed system. End users configure and submit application execution requests via a mobile app, which has the functions of data collection, task execution and application server communication. We installed a Spring Boot server in cloud and edge machines, and implemented all services for offloading optimisation, service queuing, task execution, data communication and file management. Finally, the offloading framework was validated with respect to real application execution. To achieve optimal, reasonable and sufficient validation performance, a profiling procedure was applied to generate accurate estimates of energy consumption related to processing on the mobile device and offloading data to remote servers. The proposed framework was validated on hybrid MCC and MECC systems. Results show low validation error, averaging less than 10% for the offloading optimisation parameters.

7.2 Future Directions

This thesis addressed the problem of data-intensive application offloading on mobile-aware computing systems, and proposed and validated several novel solutions. However, some research challenges remain open for future exploration.

7.2.1 Optimisation problems

Computation offloading is the process of migrating complex tasks to powerful resources to overcome the limitations of mobile devices [122]. Many techniques and approaches have been proposed to solve the offloading optimisation problem

in various contexts and scenarios. Most of these optimisation techniques have focused on achieving optimisation objectives though adopting high performance resource and communication models. However, with advances mobile research toward streaming services [123], content delivery applications, social sensing [124] and privacy-awareness enabled applications, offloading optimisation needs to be taken to a higher level using workload-based techniques. In this thesis workload-based offloading was studied using generic application scenarios. More research is required to develop advanced offloading mechanisms that depend on application-specific modelling strategies and sophisticated optimisation techniques at large-scale with machine learning an artificial intelligence technologies [125, 126].

7.2.2 Context awareness

Context awareness has been broadly studied in the scope of dynamic offloading to get benefit from the timely changing on some environmental features such as user location and system states to propose optimised offloading plans. Handling context awareness improves the performance and the applicability of an offloading technique, since computation offloading does not always yield useful outcomes [127]. With respect to data-intensive applications, offloading planning is sensitive to context information like network quality and resource availability. For example, in streaming applications, network quality plays a vital role in offloading performance and accordingly application service quality. For such applications, applying dynamic offloading schemes can be more beneficial. Thus, there is considerable scope for research into dynamic offloading with respect to sensitive context aware applications and with the support of cutting-edge technologies like edge computing, 5G and software-defined networking (SDN) [128, 129].

7.2.3 Reliable Computation Offloading

The vast majority of computation offloading techniques have seek to provide optimisation solutions for latency-sensitive and energy-aware offloading problems.

An offloading process is subject to failure for many reasons, including resource unavailability and inadequate network conditions [130]. Therefore, it is critical for offloading systems to guarantee reliable offloading execution, which refers to continuous and successful processing cycles for the desired application. For some applications, reliability is a critical factor. For instance, in Internet of vehicles (IoV) applications like autonomous driving, interruptions of communication links and processing node failure are inevitable [131]. As reliable offloading aims to ensure a high probability of successful execution, techniques like joint computation offloading, dynamic task allocation and edge-enabled SDN workload distribution are promising future research directions for reliable latency-sensitive applications [132]. However, with increasing IoT and IoV adoption in real-time applications, researchers need to provide efficient solutions to the problem of moving toward complex computation scenarios and structures to achieve reliability and heterogeneity constraints.

7.2.4 Security and privacy

Data exchange is a fundamental aspect of computation offloading systems. Ensuring data privacy and security as part of the offloading process is essential due to the engagement of users' personal devices for process activation, data collection and related tasks [133]. There is huge investment in privacy and security research, mostly focused on incorporating technologies like blockchain and SDN to sustain secure offloading processing pipelines closer to user devices [134]. Edge-based computing systems are relevant to secure offloading scenarios because they can support closed computation cycles by migrating sensitive data to the cloud. Privacy, integrity, joint authentication, and anonymity are the main parameters considered while formulating security-enabled offloading systems. The development of these applications is mostly complex, and to date few researchers have proposed workable security-based offloading applications and frameworks. [130, 135, 136].

7.3 Conclusions

Computation offloading is a common approach to resolving issues of local mobile execution related to limited computation and storage capacity and dependence on a single source of energy. However, with the emergence of new applications which involve processing large data files or sending continuous data streams for remote execution, and which have increasingly large requirements with respect to both computation and data insensitivity, offloading processes and application execution planning are becoming even more complex. This thesis proposed algorithms and techniques for optimising the execution of data-intensive applications through adopting data-oriented application modelling, which reflects the contribution of data parameter to the optimisation objectives of time, cost and energy. Moreover, the thesis provides offloading optimisation techniques which are aligned with the application complexity and structure. Several experiments were undertaken for offloading optimisation on different computing systems. Finally, a comprehensive analysis of the behaviour of data-intensive offloading techniques is presented.

Bibliography

[1] Cisco Visual Networking Index. global mobile data traffic forecast update, 2017–2022. *White paper*, 2019.

[2] BV RamaKrishna and B Sushma. Mobile cloud diabetic control system. *International Journal of Computer Science and Mobile Computing*, 6(7): 121–127, 2017.

[3] Hind Bangui, Said Rakrak, and Said Raghay. External sources for mobile computing: The state-of-the-art, challenges, and future research. In *Cloud Technologies and Applications (CloudTech), 2015 International Conference on*, pages 1–8. IEEE, 2015.

[4] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[5] Amina Rashid and Javed Parvez. Mobile cloud computing: A survey of emerging issues and future trends. *International Journal of Engineering Science and Technology*, 6(6):295, 2014.

[6] Mazliza Othman, Sajjad Ahmad Madani, Samee Ullah Khan, et al. A survey of mobile cloud computing application models. *IEEE communications surveys & tutorials*, 16(1):393–413, 2013.

[7] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[8] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993.

[9] George H. Forman and John Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, 1994.

[10] Mahadev Satyanarayanan. Fundamental challenges in mobile computing. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–7, 1996.

[11] Shunxing Chen and Linfeng Yang. WAP (wireless application protocol). *Helsinki University of Technology*, 1998.

[12] BO Eke. WAP, HTTP and HTML5 Web Socket Architecture Analysis in Contemporary Mobile App Development. *International Journal of Computer Applications Technology and Research*, 4(9):655–663, 2015.

[13] Fei Zhang, Guangming Liu, Bo Zhao, Xiaoming Fu, and Ramin Yahyapour. Reducing the network overhead of user mobility–induced virtual machine migration in mobile edge computing. *Software: Practice and Experience*, 49 (4):673–693, 2019.

[14] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.

[15] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53 (4):50–58, 2010.

[16] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *Communications of the ACM*, 39(1):50–55.

[17] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10. Ieee, 2008.

[18] Yating Wang, Ray Chen, and Ding-Chau Wang. A survey of mobile cloud computing applications: perspectives and challenges. *Wireless Personal Communications*, 80(4):1607–1623, 2015.

[19] Cisco Visual Networking Index Cisco. Global mobile data traffic forecast update, 2013-2018 (white paper, 2014).

[20] Arif Ahmed, Abadhan Saumya Sabyasachi, and Esha Barlaskar. Study on various qos issue in mobile cloud computing and future direction. In *8th International Conference on Communication Network*, page 48–255, 2014.

[21] Saeid Abolfazli, Zohreh Sanaei, Abdullah Gani, Feng Xia, and Laurence T Yang. Rich mobile applications: genesis, taxonomy, and open issues. *Journal of Network and Computer Applications*, 40:345–362, 2014.

[22] Verdi March, Yan Gu, Erwin Leonardi, George Goh, Markus Kirchberg, and Bu Sung Lee. μcloud: towards a new paradigm of rich mobile applications. *Procedia Computer Science*, 5:618–624, 2011.

[23] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Infocom, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[24] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[25] Bowen Zhou, Amir Vahid Dastjerdi, Rodrigo N Calheiros, Satish Narayana Srirama, and Rajkumar Buyya. A context sensitive offloading scheme for mobile cloud computing service. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 869–876. IEEE, 2015.

[26] Pulkit Gupta. Evolvement of mobile generations: 1g to 5g. *International Journal for Technological Research in Engineering*, 1:152–157, 2013.

[27] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611, 2013.

[28] Huber Flores and Satish Narayana Srirama. Mobile cloud middleware. *Journal of Systems and Software*, 92:82–94, 2014.

[29] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.

[30] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5): 637–646, 2016.

[31] Pramod Abichandani, William Fligor, and Eli Fromm. A cloud enabled virtual reality based pedagogical ecosystem for wind energy education. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–7. IEEE, 2014.

[32] Athar Ali Khan, Mubashir Husain Rehmani, and Martin Reisslein. Cognitive radio for smart grids: Survey of architectures, spectrum sensing mechanisms, and networking protocols. *IEEE Communications Surveys & Tutorials*, 18 (1):860–898, 2015.

[33] Ejaz Ahmed, Ibrar Yaqoob, Abdullah Gani, Muhammad Imran, and Mohsen Guizani. Internet-of-things-based smart environments: state of the art, taxonomy, and open research challenges. *IEEE Wireless Communications*, 23 (5):10–16, 2016.

[34] Milan Patel, Brian Naughton, Caroline Chan, Nurit Sprecher, Sadayuki Abeta, Adrian Neal, et al. Mobile-edge computing introductory technical white paper. *White paper, mobile-edge computing (MEC) industry initiative*, pages 1089–7801, 2014.

[35] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.

[36] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[37] Pengfei Wang, Chao Yao, Zijie Zheng, Guangyu Sun, and Lingyang Song. Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems. *IEEE Internet of Things Journal*, 6(2):2872–2884, 2018.

[38] MECISG ETSI. Mobile edge computing (mec); framework and reference architecture. *ETSI, DGS MEC*, 3, 2016.

[39] Zhuo Li, Xu Zhou, and Yifang Qin. A survey of mobile edge computing in the industrial internet. In *2019 7th International Conference on Information, Communication and Networks (ICICN)*, pages 94–98. IEEE, 2019.

[40] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[41] Jinke Ren, Guanding Yu, Yinghui He, and Geoffrey Ye Li. Collaborative cloud and edge computing for latency minimization. *IEEE Transactions on Vehicular Technology*, 68(5):5031–5044, 2019.

[42] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[43] Rajesh Balan, Jason Flinn, Mahadev Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European*, pages 87–92, 2002.

[44] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010.

[45] Andreas Klein, Christian Mannweiler, Joerg Schneider, and Hans D Schotten. Access schemes for mobile cloud computing. In *2010 Eleventh International Conference on Mobile Data Management*, pages 387–392. IEEE, 2010.

[46] Xiaohui Gu, Klara Nahrstedt, Alan Messer, Ira Greenberg, and Dejan Milojicic. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing*, 3(3):66–73, 2004.

[47] Huber Flores, Satish Narayana Srirama, and Carlos Paniagua. Towards mobile cloud applications. *International Journal of Pervasive Computing and Communications*, 8(4):344–367, 2012.

[48] Saeid Abolfazli, Zohreh Sanaei, Abdullah Gani, Feng Xia, and Wei-Ming Lin. Rmcc: Restful mobile cloud computing framework for exploiting adjacent service-based mobile cloudlets. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 793–798. IEEE, 2014.

[49] MG Siegler. Apple's massive new data center set to host nuance tech, 2017.

[50] Emad Elwany and Siamak Shakeri. Enhancing cortana user experience using machine learning. *Recall*, 55(54.61):24–24, 2014.

[51] Xiaoming Nan, Yifeng He, and Ling Guan. Optimal resource allocation for multimedia cloud based on queuing model. In *Multimedia signal processing (MMSP), 2011 IEEE 13th international workshop on*, pages 1–6. IEEE, 2011.

[52] Mirco Franzago, Henry Muccini, and Ivano Malavolta. Towards a collaborative framework for the design and development of data-intensive mobile applications. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, pages 58–61, 2014.

[53] Mohammed Anowarul Hassan and Songqing Chen. An investigation of different computing sources for mobile application outsourcing on the road.

In *International Conference on Mobile Wireless Middleware, Operating Systems, and Applications*, pages 153–166. Springer, 2011.

[54] Mohammed A Hassan, Kshitiz Bhattarai, and Songqing Chen. vups: Virtually unifying personal storage for fast and pervasive data accesses. In *International Conference on Mobile Computing, Applications, and Services*, pages 186–204. Springer, 2012.

[55] Aaron Carroll, Gernot Heiser, et al. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14, pages 21–21. Boston, MA, 2010.

[56] Azzedine Boukerche, Shichao Guan, and Robson E De Grande. Sustainable offloading in mobile cloud computing: Algorithmic design and implementation. *ACM Computing Surveys (CSUR)*, 52(1):1–37, 2019.

[57] Dong Huang, Ping Wang, and Dusit Niyato. A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications*, 11 (6):1991–1995, 2012.

[58] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, D Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28 (13):2009, 2009.

[59] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[60] M Reza Rahimi, Nalini Venkatasubramanian, Sharad Mehrotra, and Athanasios V Vasilakos. Mapcloud: mobile applications on an elastic and scalable 2-tier cloud architecture. In *Proceedings of the 2012 IEEE/ACM fifth international conference on utility and cloud computing*, pages 83–90. IEEE Computer Society, 2012.

[61] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. Calling the cloud: enabling mobile phones as interfaces to cloud applications. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, page 5. Springer-Verlag New York, Inc., 2009.

[62] Ting-Yi Lin, Ting-An Lin, Cheng-Hsin Hsu, and Chung-Ta King. Context-aware decision engine for mobile cloud offloading. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2013 IEEE*, pages 111–116. IEEE, 2013.

[63] Farshad A Samimi, Philip K McKinley, and S Masoud Sadjadi. Mobile service clouds: A self-managing infrastructure for autonomic mobile computing services. In *IEEE International Workshop on Self-Managed Networks, Systems, and Services*, pages 130–141. Springer, 2006.

[64] Inés Sittón-Candanedo, Ricardo S Alonso, Juan M Corchado, Sara Rodríguez-González, and Roberto Casado-Vara. A review of edge computing reference architectures and a new global edge proposal. *Future Generation Computer Systems*, 99:278–294, 2019.

[65] M Tseng, T Canaran, and L Canaran. Introduction to edge computing in iiot. *Industrial Internet Consortium (IIC) White Paper, Tech. Rep*, 2018.

[66] Yuyi Mao, Jun Zhang, and Khaled B Letaief. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, 2016.

[67] Nur Idawati Md Enzai and Maolin Tang. A heuristic algorithm for multi-site computation offloading in mobile cloud computing. *Procedia Computer Science*, 80:1232–1241, 2016.

[68] Yaser Jararweh, Mahmoud Al-Ayyoub, Muneera Al-Quraan, A Tawalbeh Lo'ai, and Elhadj Benkhelifa. Delay-aware power optimization model for mobile edge computing systems. *Personal and Ubiquitous Computing*, 21 (6):1067–1077, 2017.

[69] Mohammad Goudarzi, Mehran Zamani, and Abolfazl Toroghi Haghighat. A fast hybrid multi-site computation offloading for mobile cloud computing. *Journal of Network and Computer Applications*, 80:219–231, 2017.

[70] Min Chen and Yixue Hao. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597, 2018.

[71] Majid Altamimi, Rajesh Palit, Kshirasagar Naik, and Amiya Nayak. Energy-as-a-service (eaas): On the efficacy of multimedia cloud computing to save smartphone energy. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 764–771. IEEE, 2012.

[72] Muhammad Shiraz, Saeid Abolfazli, Zohreh Sanaei, and Abdullah Gani. A study on virtual machine deployment for application outsourcing in mobile cloud computing. *The Journal of Supercomputing*, 63(3):946–964, 2013.

[73] Xinwen Zhang, Sangoh Jeong, Anugeetha Kunjithapatham, and Simon Gibbs. Towards an elastic application model for augmenting computing capabilities of mobile platforms. In *International Conference on Mobile Wireless Middleware, Operating Systems, and Applications*, pages 161–174. Springer, 2010.

[74] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 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, 2010.

[75] Jason Flinn, Dushyanth Narayanan, and Mahadev Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proceedings Eighth Workshop on Hot Topics in Operating Systems*, pages 61–66. IEEE, 2001.

[76] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: a computation offloading framework for smartphones. In *International*

Conference on Mobile Computing, Applications, and Services, pages 59–79. Springer, 2010.

[77] Yan Ding, Gaochao Xu, Chunyi Wu, Liang Hu, Yunan Zhai, and Jia Zhao. Explore virtual machine deployment to mobile cloud computing for multi-tenancy and energy conservation in wireless network. *Cluster Computing*, 20(4):3263–3274, 2017.

[78] Yi-Hsuan Kao, Bhaskar Krishnamachari, Moo-Ryong Ra, and Fan Bai. Hermes: Latency optimal task assignment for resource-constrained mobile computing. *IEEE Transactions on Mobile Computing*, 16(11):3056–3069, 2017.

[79] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. Enacloud: An energy-saving application live placement approach for cloud computing environments. In *2009 IEEE International Conference on Cloud Computing*, pages 17–24. IEEE, 2009.

[80] Mati B Terefe, Heezin Lee, Nojung Heo, Geoffrey C Fox, and Sangyoon Oh. Energy-efficient multisite offloading policy using markov decision process for mobile cloud computing. *Pervasive and Mobile Computing*, 27:75–89, 2016.

[81] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2015.

[82] Lei Yang, Jiannong Cao, Hui Cheng, and Yusheng Ji. Multi-user computation partitioning for latency sensitive mobile cloud applications. *IEEE Transactions on Computers*, 64(8):2253–2266, 2014.

[83] Wenlu Hu, Ying Gao, Kiryong Ha, Junjue Wang, Brandon Amos, Zhuo Chen, Padmanabhan Pillai, and Mahadev Satyanarayanan. Quantifying the impact of edge computing on mobile applications. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 1–8, 2016.

[84] Weiwen Zhang, Yonggang Wen, and Dapeng Oliver Wu. Collaborative task execution in mobile cloud computing under a stochastic wireless channel. *IEEE Transactions on Wireless Communications*, 14(1):81–93, 2014.

[85] Valeria Cardellini, Vittoria De Nitto Personé, Valerio Di Valerio, Francisco Facchinei, Vincenzo Grassi, Francesco Lo Presti, and Veronica Piccialli. A game-theoretic approach to computation offloading in mobile cloud computing. *Mathematical Programming*, 157(2):421–449, 2016.

[86] Quyuan Wang, Songtao Guo, Jiadi Liu, and Yuanyuan Yang. Energy-efficient computation offloading and resource allocation for delay-sensitive mobile edge computing. *Sustainable Computing: Informatics and Systems*, 21:154–164, 2019.

[87] M Reza Rahimi, Nalini Venkatasubramanian, and Athanasios V Vasilakos. Music: Mobility-aware optimal service allocation in mobile cloud computing. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 75–82. IEEE, 2013.

[88] Anind K Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[89] Hyun Jung La and Soo Dong Kim. A conceptual framework for provisioning context-aware mobile cloud services. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 466–473. IEEE, 2010.

[90] Bowen Zhou, Amir Vahid Dastjerdi, Rodrigo N Calheiros, Satish Narayana Srirama, and Rajkumar Buyya. mcloud: A context-aware offloading framework for heterogeneous mobile cloud. *IEEE Transactions on Services Computing*, 10(5):797–810, 2015.

[91] Hyunseok Chang, Adiseshu Hari, Sarit Mukherjee, and TV Lakshman. Bringing the cloud to the edge. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 346–351. IEEE, 2014.

[92] Vitor Barbosa C Souza, Wilson Ramírez, Xavier Masip-Bruin, Eva Marín-Tordera, G Ren, and Ghazal Tashakor. Handling service allocation in combined fog-cloud scenarios. In *2016 IEEE International Conference on Communications*, pages 1–5. IEEE, 2016.

[93] Claudio Feijóo, José Luis Gómez-Barroso, and Sergio Ramos. Implications of data-intensive applications for next generation mobile networks. 2014.

[94] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang, and Tao Li. Edge-oriented computing paradigms: A survey on architecture design and system management. *ACM Computing Surveys (CSUR)*, 51(2):1–34, 2018.

[95] Lei Jiao, Roy Friedman, Xiaoming Fu, Stefano Secci, Zbigniew Smoreda, and Hannes Tschofenig. Cloud-based computation offloading for mobile devices: State of the art, challenges and opportunities. In *2013 Future Network & Mobile Summit*, pages 1–11. IEEE, 2013.

[96] Tianchu Zhao, Sheng Zhou, Xueying Guo, Yun Zhao, and Zhisheng Niu. A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2015.

[97] Thai T Vu, Diep N Nguyen, Dinh Thai Hoang, and Eryk Dutkiewicz. Optimal task offloading and resource allocation for fog computing. *arXiv preprint arXiv:1906.03567*, 2019.

[98] Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, and Muhammad Shiraz. Sami: Service-based arbitrated multi-tier infrastructure for mobile cloud computing. In *Communications in China Workshops (ICCC), 2012 1st IEEE International Conference on*, pages 14–19. IEEE, 2012.

[99] Saeid Abolfazli, Abdullah Gani, and Min Chen. Hmcc: A hybrid mobile cloud computing framework exploiting heterogeneous resources. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 157–162. IEEE, 2015.

[100] Walfredo Cirne, Daniel Paranhos, Lauro Costa, Elizeu Santos-Neto, Francisco Brasileiro, Jacques Sauve, Fabrício AB Silva, Carla O Barros, and Cirano Silveira. Running bag-of-tasks applications on computational grids: The mygrid approach. In *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pages 407–416. IEEE, 2003.

[101] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

[102] John E Shore. Information theoretic approximations for m/g/1 and g/g/1 queuing systems. *Acta Informatica*, 17(1):43–61, 1982.

[103] John DC Little. A proof for the queuing formula: L= λ w. *Operations Research*, 9(3):383–387, 1961.

[104] Cosimo Anglano and Massimo Canonico. Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8. IEEE, 2008.

[105] Z Yang. Powertutor-a power monitor for android-based mobile platforms. *EECS, University of Michigan, retrieved September*, 2:19, 2012.

[106] Inés Sittón and Sara Rodríguez. Pattern extraction for the design of predictive models in industry 4.0. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 258–261. Springer, 2017.

[107] Shih-Hao Hung, Chi-Sheng Shih, Jeng-Peng Shieh, Chen-Pang Lee, and Yi-Hsiang Huang. Executing mobile applications on the cloud: Framework and issues. *Computers & Mathematics with Applications*, 63(2):573–587, 2012.

[108] William G Marchal. An approximate formula for waiting time in single server queues. *AIIE transactions*, 8(4):473–474, 1976.

[109] George Terzopoulos and Helen D Karatza. Power-aware bag-of-tasks scheduling on heterogeneous platforms. *Cluster Computing*, 19(2):615–631, 2016.

[110] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *Siam Review*, 57(1):3–57, 2015.

[111] Luca Urbanucci. Limits and potentials of mixed integer linear programming methods for optimization of polygeneration energy systems. *Energy Procedia*, 148:1199–1205, 2018.

[112] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[113] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[114] Mohammad Alkhalaileh, Rodrigo N Calheiros, Quang Vinh Nguyen, and Bahman Javadi. Dynamic resource allocation in hybrid mobile cloud computing for data-intensive applications. In *International Conference on Green, Pervasive, and Cloud Computing*, pages 176–191. Springer, 2019.

[115] Mainak Adhikari, Tarachand Amgoth, and Satish Narayana Srirama. A survey on scheduling strategies for workflows in cloud environment and emerging trends. *ACM Computing Surveys*, 52(4):1–36, 2019.

[116] Chi-Tsun Cheng, Nuwan Ganganath, and Kai-Yin Fok. Concurrent data collection trees for iot applications. *IEEE Transactions on Industrial Informatics*, 13(2):793–799, 2016.

[117] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science*, pages 1–10. IEEE, 2008.

[118] Mohammad Alkhalaileh, Rodrigo N Calheiros, Quang Vinh Nguyen, and Bahman Javadi. Data-intensive application scheduling on mobile edge cloud

computing. *Journal of Network and Computer Applications*, page 102735, 2020.

[119] Marco V Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *2013 Proceedings Ieee Infocom*, pages 1285–1293. IEEE, 2013.

[120] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 1–14, 2011.

[121] Gary Mak. Spring mvc framework. In *Spring Recipes*, pages 321–393. Springer, 2008.

[122] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246, 2001.

[123] Anas Toma and Jian-Jia Chen. Computation offloading for real-time systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1650–1651, 2013.

[124] Peng-Yong Kong. Computation and sensor offloading for cloud-based infrastructure-assisted autonomous vehicles. *IEEE Systems Journal*, 2020.

[125] Junaid Shuja, Kashif Bilal, Eisa Alanazi, Waleed Alasmary, and Abdulaziz Alashaikh. Applying machine learning techniques for caching in edge networks: A comprehensive survey. *arXiv preprint arXiv:2006.16864*, 2020.

[126] Bo Yang, Xuelin Cao, Joshua Bassey, Xiangfang Li, and Lijun Qian. Computation offloading in multi-access edge computing: A multi-task learning approach. *IEEE Transactions on Mobile Computing*, 2020.

[127] Mazliza Othman, Feng Xia, Abdul Nasir Khan, et al. Context-aware mobile cloud computing and its challenges. *IEEE Cloud Computing*, 2(3):42–49, 2015.

[128] Tian Wang, Yuzhu Liang, Yilin Zhang, Muhammad Arif, Jin Wang, Qun Jin, et al. An intelligent dynamic offloading from cloud to edge for smart iot systems with big data. *IEEE Transactions on Network Science and Engineering*, 2020.

[129] Qi Zhang, Lin Gui, Fen Hou, Jiacheng Chen, Shichao Zhu, and Feng Tian. Dynamic task offloading and resource allocation for mobile-edge computing in dense cloud ran. *IEEE Internet of Things Journal*, 7(4):3282–3299, 2020.

[130] Quang-Huy Nguyen and Falko Dressler. A smartphone perspective on computation offloading–a survey. *Computer Communications*, 2020.

[131] Shi Yang. A task offloading solution for internet of vehicles using combination auction matching model based on mobile edge computing. *IEEE Access*, 8:53261–53273, 2020.

[132] Xiangwang Hou, Zhiyuan Ren, Jingjing Wang, Wenchi Cheng, Yong Ren, Kwang-Cheng Chen, and Hailin Zhang. Reliable computation offloading for edge computing-enabled software-defined iov. *IEEE Internet of Things Journal*, 2020.

[133] Yuanfan Yao, Ziyu Wang, and Pan Zhou. Privacy-preserving and energy efficient task offloading for collaborative mobile computing in iot: An admm approach. *Computers & Security*, page 101886, 2020.

[134] Darshan Vishwasrao Medhane, Arun Kumar Sangaiah, M Shamim Hossain, Ghulam Muhammad, and Jin Wang. Blockchain-enabled distributed security framework for next generation iot: An edge-cloud and software defined network integrated approach. *IEEE Internet of Things Journal*, 2020.

[135] Anju Rana and Farshid Hajati. A survey on privacy and security in mobile cloud computing. In *Workshops of the International Conference on Advanced Information Networking and Applications*, pages 1065–1076. Springer, 2020.

[136] Shuai Yu, Xu Chen, Lei Yang, Di Wu, Mehdi Bennis, and Junshan Zhang. Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading. *IEEE Wireless Communications*, 27(1):92–99, 2020.