Code Offloading and Resource Management Algorithms for Heterogeneous Mobile Clouds

Bowen Zhou

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

January 2018

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

Code Offloading and Resource Management Algorithms for Heterogeneous Mobile Clouds

Bowen Zhou
Principal Supervisor: Prof. Rajkumar Buyya

Abstract

The recent innovation and development of software and hardware on mobile devices such as smartphones and tablets have made them evolve as the primary tool of our digital life. Mobile device users have increased their demands on more PC-like user experiences such as mobile gaming, augmented reality, and mobile version of legacy PC applications. However, the battery lifetime remains as a weakness due to the advanced display, camera and sensors that drain the battery quickly. Since the battery technology seems unlikely to have a significant improvement in the foreseen future, other solutions are needed.

The heterogeneous mobile cloud paradigm emerges as a promising solution. It defines a wireless, shared computing resource environment that consists of ad-hoc connected mobile devices, nearby form-factor servers (cloudlets), and cloud computing services. Mobile devices can leverage the shared resource environment to offload their computation intensive tasks in order to conserve battery lifetime and accelerate application performances. However, with such a loosely coupled and mobile device dominating network, new challenges and problems emerge such as how to enable task offloading in such environment, how to achieve minimum time and energy consumption through task offloading and scheduling, how to maintain the service reliability and recover from failures, and how to incentivize ad-hoc mobile users to use mobile cloud offloading services.

This thesis studies algorithms and technologies to solve these problems and provides a task offloading framework for the heterogeneous mobile cloud service. It extended the state-of-the-art by making the following key contributions:

1. A system architecture for heterogeneous mobile cloud services and a system framework implemented on Android platform to enable the mobile cloud offloading service.

2. A context-aware offloading algorithm for individual mobile devices to make task offloading decisions based on the context changes in the heterogeneous mobile cloud network.

3. An optimal offline algorithm and an online algorithm based on the ski-rental framework with near-optimal performance guarantee for providing task offloading and scheduling decisions on minimizing overall task execution time, considering the load balancing of all devices in the heterogeneous mobile cloud network as well as the unique constraints such as battery limit and offloading enhancement.

4. A group-based fault tolerant mechanism for improving the service reliability of heterogeneous mobile cloud offloading. It classifies mobile devices into groups

based on its processing capacity. mobility, and reliability. Different fault tolerant technologies such as checkpointing and replication are devised adaptively based on the task offloading schedules and the specific group of machines it's offloaded.

5. A reverse auction based incentive mechanism and an optimal offline model for increasing the participation and utilization of the proposed mobile cloud offloading services. The proposed algorithm guarantees computational efficiency, truthfulness, individual rationality and near-optimal auction results.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Bowen Zhou, 25 January 2018

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my principal supervisor Prof. Rajkumar Buyya. Throughout my PhD candidature, Prof. Buyya has provided me the continuous support with his patience, enthusiasm, and immense knowledge. His guidance has motivated me to carry out the research works in this thesis, without which this could not ever happen. I also thank my external supervisor Dr. Satish Srirama for his advice and guidance on my research. I thank them for their contributions on the research and writing of this thesis.

Besides my supervisors, I would like to thank the chair of my committee, Prof. Lars Kulik for his kindness and advice to help me progress along my PhD candidature. I also want to thank Dr. Rodrigo N. Calheiros and Dr. Amir Vahid Dastjerdi for their invaluable advices and discussions during early part of my PhD studies.

I also want to take the opportunity to thank some of the current and past members of the CLOUDS laboratory at the University of Melbourne: Dr. Adel Nadjaran Toosi, Dr. Deepak Poola, Dr. Nikolay Grozev, Dr. Maria Rodriguez, Dr. Chenhao Qu, Dr. Atefeh Khosravi, Dr. Yaser Mansouri, Dr. Sukhpal Singh Gill, Jungmin Son, Safiollah Heidari, Xunyun Liu, Caesar Wu, Minxian Xu, Sara Kardani Moghaddam, Muhammad H. Hilman, Redowan Mahmud, Muhammed Tawfiqul Islam, Shashikant Ilager, TianZhang He, Tiago Justino, and Diana Barreto for their help and friendships.

I thank the University of Melbourne for the scholarship, research training support and facilities to let me pursue my PhD degree. My sincere thank also goes to the staff at the School including Rhonda Smithies, Julie Ireland, Madalain Dolic for their support.

Last but not least, I want to thank my parents for the emotional support and endless love throughout this journey and always believing in me. Their comfort and encouragement have carried me on a long way through the difficulties in both my PhD candidature and life.

Bowen Zhou
Melbourne, Australia
January 2018

Preface

The thesis research has been carried out in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2 - 6 and are based on the following publications:

- **Bowen Zhou** and Rajkumar Buyya, "Augmentation Techniques for Mobile Cloud Computing: A Taxonomy, Survey, and Future Directions," *ACM Computing Surveys*, Volume 51, No. 1, Article No. 13, Pages: 1-38, ISSN 0360-0300, ACM Press, New York, USA, January 2018.

- **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 Proceedings of IEEE 8th International Conference on Cloud Computing (CLOUD)*, New York City, NY, USA, 27 June-2 July, 2015, Pages: 869-876.

- **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*, Volume 10, Issue 5, September/October 2017, Pages: 797 - 810.

- **Bowen Zhou**, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya, "An Online Algorithm for Task Offloading in Heterogeneous Mobile Clouds," *ACM Transactions on Internet Technology*, Volume 18 Issue 2, Article No. 23, March 2018.

- **Bowen Zhou** and Rajkumar Buyya, "A Group-Based Fault Tolerant Mechanism for Heterogeneous Mobile Clouds," *in Proceedings of the 14th EAI International Confer-*

ence on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2017), Melbourne, VIC, Australia, November 7 - 10, 2017.

- textbfBowen Zhou, Satish Srirama, and Rajkumar Buyya, "An Auction-Based Incentive Mechanism for Heterogeneous Mobile Clouds," *Journal of Parallel and Distributed Computing*, 2018 (Under review).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

THE recent innovation and development on mobile devices such as smartphones and tablets have made the smart devices evolve as the primary tool in our digital life. According to some recent mobile industry reports [49, 50], mobile device usage has doubled in the past three years, and has represented 70% of digital media minutes by the end of 2016. However, the limited computing capacity of the embedded hardware can not fulfil the mobile users' growing expectations of applications. Moreover, the battery lifetime remains as a weakness of the mobile devices due to the high energy consumption from display, camera, sensors, etc. that end up draining the battery quickly. On the other hand, the battery technology seems unlikely to have a significant improvement in the foreseen future to keep up with the rest of the mobile hardware.

Cloud computing has gained popularity for industries as solutions for running their web services. It enables end users to run a broad range of applications including scientific workflows, data analytics, streaming services, etc. on the cloud services. NIST defines cloud computing with a three-tier service model [159]. The Infrastructure-as-a-Service (IaaS) model at bottom provides resources such as compute, storage and network. On top of IaaS model, there is Platform-as-a-Service (PaaS) that provides application developing and management environment. The top tier is Software-as-a-Service (SaaS) model, which provides services through web applications to end users. In this thesis, the focus is on the IaaS model.

Mobile cloud computing emerges as a promising solution to the issues of modern smart mobile devices. At the early stage, mobile cloud computing is a onefold computing paradigm between a mobile device and a cloud service that enables the mobile device to outsource the computation intensive tasks onto cloud computing resources via wire-

less networks for execution. However, as the outsourcing data from mobile applications increases rapidly, and the mobile devices become more powerful, network bottleneck and lack of performance improvement degrade the effect of the mobile cloud computing.

Therefore, this thesis focuses on the new paradigm named heterogeneous mobile clouds that aims to avert the network bottleneck and ensure adaptation to different mobile cloud environments. The new paradigm introduced the mobile ad-hoc network [218] and cloudlet [200] into the original mobile cloud computing. Figure 1.1 shows an example of the HMC (Heterogeneous Mobile Cloud) where the mobile devices in a local area form a wireless ad-hoc network through short-range wireless network such as WiFi, Bluetooth and WiFi-direct. In addition, some users also own cloudlets that are publicly connectible for other mobile devices via WiFi (e.g., a café owner), and some users have subscribed to the application's cloud service with dedicated instances running in the cloud. HMC can be applied with different availability of network mediums. The different types of resources form a shared resource pool that enables mobile cloud offloading services. As can be observed, the HMC environment is a dynamic computing environment that contains heterogeneous machines and tasks, intermittent wireless networks, and context changes.

In the remainder of this chapter, motivation, challenges, research problems of this thesis, evaluation methodology, thesis contributions, and the thesis organization are discussed.

1.1 Motivation and Challenges

1.1.1 Motivation

Mobile devices have already seen a significant improvement with more powerful and functional hardware such as multi-core processors, mobile graphic cards, and memory. Nevertheless, realizing mobile cloud computing can bring numerous benefits to both mobile device users and cloud service providers. From the user's perspective, user experience on mobile applications and functions of mobile devices can be improved by introducing mobile cloud computing. For cloud service providers, mobile clouds enable a large user community to use their services. Additionally, providers can apply machine

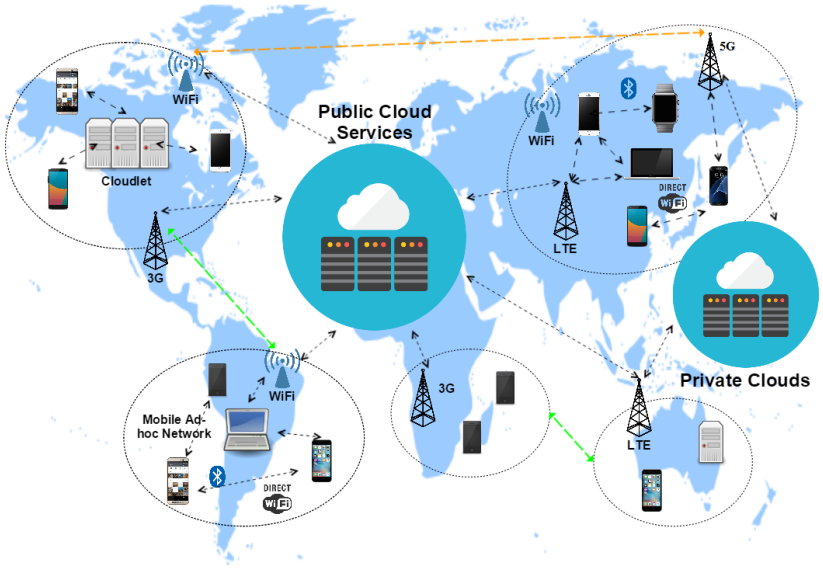


Figure 1.1: An overview of the heterogeneous mobile cloud environment

learning mechanisms on the service data of mobile device users to further provide more customized cloud services.

Computing capability

Despite hardware improvements seen in mobile devices in recent years, the slow processing speed and low RAM still hinder mobile devices from providing experience rich applications to end users [199]. Since mobile devices have been involved as part of people's daily activities such as social networking, office work, and gaming, they are expected to have PC approximate computing capacities. Mobile cloud computing provides the opportunities for developers to overcome this gap and enrich their applications by outsourcing computing-intensive tasks to cloud resources. Moreover, by migrating the computation to other computing resources, mobile devices are released to complete light tasks which eventually will improve the fluency of mobile devices.

Energy efficiency

One of the most urgent technical challenges for mobile devices is battery life. Currently, lithium-ion batteries used on mobile devices can only supply few hours of power for

extensive computing. As more powerful processors, bigger displays and different types of sensors are installed on mobile devices, larger energy consumption is expected. However, the battery manufacturers are only able to increase the battery capacity by 5% annually [192]. Multiple efforts have been provided by phone manufacturers such as battery saving mode that dims displays or switches off wireless interfaces when the battery level is too low. Apparently, this type of approach neither solves the problem nor enhances user experiences. Therefore, mobile cloud augmentation can be a promising alternative solution that shifts the computation from mobile devices to other resources to save energy consumption on the battery.

Seamless and collaborative mobile application

With the advent of smart mobile devices, mobile users expect continuous mobile application experiences by using mobile cloud computing services. However, the wireless mediums that carry out data migrations are the performance bottleneck for mobile cloud services due to their instability and accessibility. In the event of network infrastructure failures, public cloud services may not be available, and mobile applications running its computation intensive tasks will be interrupted. In such a case, an ad-hoc network of mobile devices as a collaborative resource pool can be an alternate solution. Another example is the mobile social network in proximity (MSNP) [31], which enables mobile device users to interact with other users nearby via peer-to-peer wireless communications and complete tasks collaboratively.

1.1.2 Challenges

The trends of utilizing heterogeneous computing resources and multiple types of wireless networks in mobile cloud augmentation bring many challenges. We present the major challenges related to techniques used in realizing mobile cloud augmentation systems.

Outsourcing computation

First, how to outsource the computation is one of the main challenges in the mobile cloud computing. Specifically, selecting the appropriate augmentation approach based on the

characteristics of heterogeneous cloud resources is a non-trivial task. For instance, code offloading is one of the common approaches for outsourcing mobile tasks to cloud virtual machine (VM) based resources, while remote procedure calling is more suitable for SOA-based task integration. A few existing works such as MAUI [52], ThinkAir [122], CloneCloud [43], MuSIC [184], Cloudlet [200] have been proposed to solve the issue, but the efficient use of heterogeneous computing resources are yet to be studied more comprehensively.

Unstable wireless communications

Due to unique characteristics of wireless communication techniques, the network throughput, signal range, and connection stability can be downgraded by environmental changes like signal interference comparing to the wired networks. Another issue of wireless networks in the mobile cloud computing environment is the channel capacity. As the number of mobile device users grows, the service level agreement (SLA) regarding network performances is at risk of being violated. Therefore, handover strategies and seamless connection in case of a network SLA violation need to be studied when developing systems for mobile cloud augmentation.

Mobile device mobility

The mobile device management is vital to the performance of the mobile cloud augmentation system. On the one hand, the mobile device movement, together with the unstable wireless networks, can cause service failures in the mobile cloud augmentation systems frequently. On the other hand, the performance of the certain types of cloud resources like mobile device cloud (i.e., mobile ad-hoc networks) depends on routing protocols that are affected by the mobile device mobility. As a result, mobility management and fault tolerance need to be considered to provide a reliable mobile cloud service. The location-based device mobility prediction under different user context is another issue that needs to be investigated.

Context-aware augmentation decision making strategies

Outsourcing the computation to cloud resources is not always beneficial because the context of the mobile cloud environment changes rapidly. As we discussed above, the device mobility and unstable wireless networks are the main reasons that cause the change of the context. Moreover, the heterogeneity of the cloud resources and mobile applications can significantly affect the performance of the augmentation. For example, in computation outsourcing such as code offloading, transferring a significant amount of data for remote execution may consume more time and energy than running it on the mobile device itself. Instead, offloading this type of computation to a nearby mobile ad-hoc network may gain benefits from parallel execution and delay-free network transmission. Hence, it is necessary to develop augmentation decision making strategies to achieve context-awareness of the mobile cloud augmentation systems.

User participation and fairness

Since heterogeneous mobile cloud services leverage the computing power of the ad-hoc network of mobile devices, each mobile device using the service should share its own computing resources for other participants to outsource. Therefore, mobile users may be discouraged from using the HMC services due to the concern of battery lifetime and privacy. Also, the HMC service should prevent malicious users from gaining extra benefits by unfair participation. As a result, it is important to design an incentive mechanism that encourage mobile users to participate in the HMC environment with fairness and rationality.

1.2 Research Problems and Objectives

This thesis focuses on the task offloading service using the heterogeneous mobile cloud environment. The target of the research is the execution cost optimization of the mobile tasks for smart mobile devices with the consideration of load balance and fault tolerance of the entire heterogeneous mobile cloud offloading service environment. In order to provide solutions to the above-mentioned challenges in respect to the thesis target, the

following research problems are defined and investigated.

- **How to identify computation intensive tasks of a mobile application and enable task offloading to HMC environment?** A mobile application commonly consists of multiple functions that work together in sequence or parallel to deliver the service. Some of the functions can only be carried out natively on its running mobile device, e.g., user inputs, while other functions can either be executed locally or on remote computing resources, e.g., photo and voice processing. Also, different functions require different computation, which affects the performance of the whole application. In addition, since the code is offloaded via wireless mediums, the time and energy cost of the data communication as well as the selection of the available wireless medium need to be considered. Therefore, in order to offload these computation intensive functions, an offloading technique is needed to fulfil these needs.

- **How to schedule the offloading tasks among the different computing resources in the HMC environment with the consideration of the application execution cost optimization?** The execution cost of a mobile task includes the execution time and the energy consumption. Since the computing resources (i.e., mobile devices and cloud resources) are heterogeneous, a mobile task requires different execution cost on different computing resources. Therefore, a task offloading and scheduling algorithm for the HMC environment should be designed to decide the task offloading locations and schedule the offloaded tasks in respect to system load balancing and cost optimization.

- **How to improve the system reliability of HMC?** Service reliability and fault tolerance are among the most challenging issues of distributed systems. Since HMC environment consists of different mobile devices and wireless networks, it is more challenging to maintain the system reliability since the conventional fault tolerant mechanisms usually focus on systems with homogeneous machines connected by wired networks. Hence, a novel, lightweight fault tolerant mechanism is required to adapt the heterogeneity of the HMC system and cooperate with the offloading logics of the system to maintain its offloading benefits.

- **How to incentivize the mobile users to participate and ensure a fair participation?** The success of the proposed HMC services depends on the participation of mobile users and their shared computing and cloud resources. Therefore, in order to provide a complete service framework of the HMC system, an incentive mechanism is of interest to provide a fair and competitive environment in which the mobile users can be encourage to participate and get benefits from sharing their redundant computing resources.

1.3 Methodology

All the approaches to the above-mentioned research problems are developed and studied following the methodology of 1) system modelling, 2) problem formulation, 3) algorithm designing, and 4) performance evaluation and numeric results analysis. The system modelling provides the abstraction of the HMC system environment, which enables the research problem to be formulated in the mathematical form based on the models. Then the solution can be proposed by designing algorithms to solve the problem formulation. Finally, the proposed algorithms are tested with either synthetic or real-world workloads and traces to prove the effectiveness on solving the research problems.

The proposed approaches including algorithms and mechanisms are evaluated using both simulations and implemented system frameworks. In order to evaluate the performance of the approaches in practice to show the feasibility, a prototype of the proposed framework is implemented on the Android platform to test the algorithms and approaches with real mobile applications including a mobile optical character recognition (OCR) application and a mobile chess game application.

The evaluation on the implemented system is restricted in small scale due to the limited number of mobile devices available for use. Therefore, simulations are used to evaluate the proposed approaches for large-scale system settings. Due to the lack of the real-world workloads for the research problems, the workloads used in the simulations of this thesis are generated by profiling task executions of different mobile applications like the OCR application. Each application is profiled on the method level to obtain parameters such as computation requirements, data size, etc. using the Android Studio. In addition,

real-world traces including CRAWDAD traces [123] and MIT Reality Mining datasets [62] are adopted to capture the mobile device movements. Corresponding simulation tools and environment are developed to conduct the simulations. Results are compared with baseline algorithms that have been previously developed in the literature.

1.4 Thesis Contributions

In order to solve the research problems, this thesis made the following **key contributions**:

1. A taxonomy and survey of the state-of-the-art computing and storage augmentation technologies for mobile device using mobile cloud computing.

2. A system framework named mCloud that enables mobile task offloading for the heterogeneous mobile cloud system.

   - The definition of the heterogeneous mobile cloud environment.
   - The design of the framework architecture and module interactions.
   - A technique based on Java Refection to enable task offloading for mobile applications
   - A context-aware decision making algorithm based on multi-criteria decision making method for providing task offloading decisions on whether to offload and the type of wireless medium used to offload.

3. An online task offloading and scheduling algorithm with the consideration of load balancing and offloading gain of all the computing resources to provide near-optimal offloading schedules for the heterogeneous mobile cloud system.

   - The definition and formulation of the mobile cloud offloading and scheduling problem (MCOSP) in heterogeneous mobile clouds to provide global optimal solutions on whether to offload a mobile task and its execution schedule among the computing resources.
   - An offline solution of MCOSP based on mixed linear programming. The offline optimal solutions are obtained using brand-and-bound optimization algorithm.

- An online scheduling algorithm based on ski-rental framework to provide competitive task offloading schedules and lightweight process for mobile devices in real-time.

4. A group-based fault tolerant mechanism, GFT-mCloud, to improve the system reliability of the HMC offloading services and maintain the offloading benefits gained from task offloading.

   - The mechanism works as an add-on module to the existing mobile cloud offloading frameworks (e.g. the proposed mCloud).
   - A machine grouping algorithm based on decision tree method to dynamically classify computing resources such as mobile devices and cloud instances to multiple groups based on different criteria and can be expended.
   - An adaptive fault tolerant algorithm that takes task offloading schedules as inputs and generates different fault tolerant policies based on the schedules and the machine groups that the task is offloaded.

5. An incentive mechanism to encourage mobile users to participate in the HMC offloading services and get rewards from sharing their redundant computing resources.

   - The offloading service is formulated as an offloading market where there are resource sellers who want to sell their redundant resources for the offloaded tasks, and resource buyers who wish to buy the resource offerings and offload their tasks.
   - A greedy reverse auction-based algorithm to allocate sellers' offerings to buyers' offloading requests, and a price determination algorithm to decide the price paid from buyers to sellers. The algorithms guarantee computation efficiency, truthfulness, and individual rationality.
   - A prototype implementation of the incentive mechanism based on the mCloud framework is developed to show the feasibility of the propose mechanism in practice.

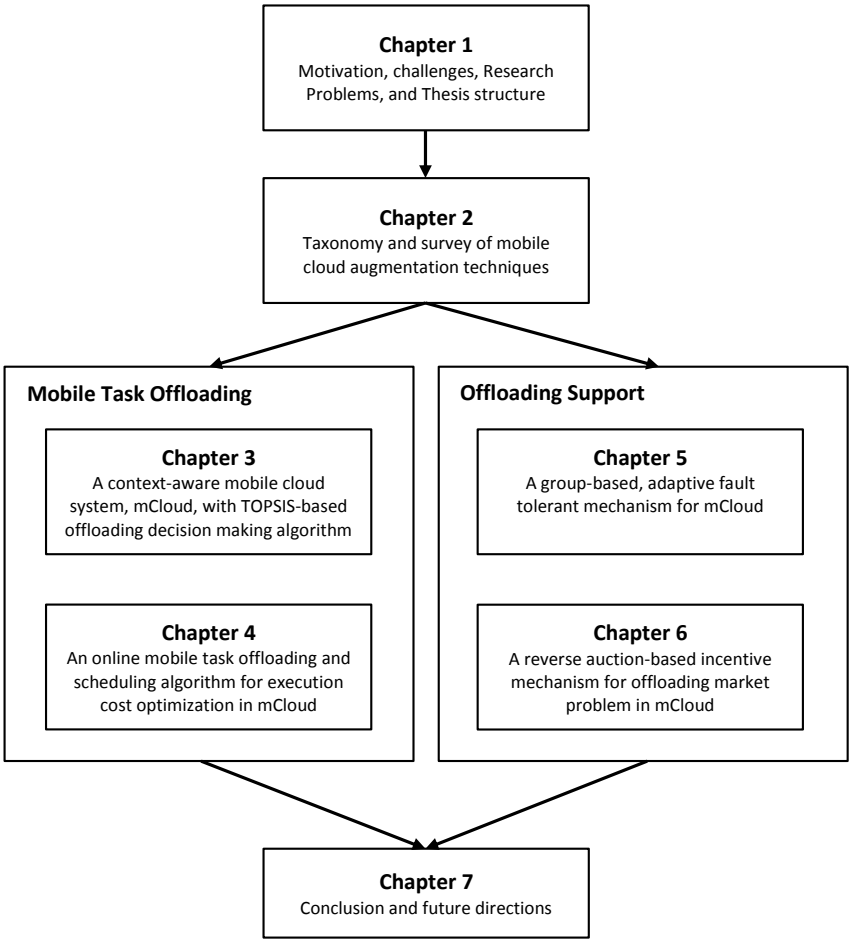


Figure 1.2: Thesis organization

1.5 Thesis Organization

The organization of the chapters in this thesis is shown in Figure 1.2. Chapter 2 discussed the state-of-the-art mobile cloud augmentation techniques with a taxonomy and survey. Chapter 3 and 4 focus on enabling the efficient task offloading services in the heterogeneous mobile cloud system. Chapter 5 and 6 focus on the supporting services including fault tolerance and incentive strategy for a complete mobile cloud offloading service. The core chapters are derived from the conference and journal publications completed during my PhD candidature. The remainder of this thesis is organized as follows:

- Chapter 2 presents a taxonomy and survey on the state-of-the-art augmentation technologies adopted in the mobile cloud computing in terms of both computing capability and mobile storage. The chapter is partially derived from the following

publication:

  - **Bowen Zhou** and Rajkumar Buyya, "Augmentation Techniques for Mobile Cloud Computing: A Taxonomy, Survey, and Future Directions," *ACM Computing Surveys*, Volume 51, No. 1, Article No. 13, Pages: 1-38, ISSN 0360-0300, ACM Press, New York, USA, January 2018.

- Chapter 3 proposed a task offloading framework along with the offloading enabling technology and a context-aware mobile-to-cloud offloading decision making algorithm. The chapter is derived from the following publications:

  - **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 Proceedings of IEEE 8th International Conference on Cloud Computing (CLOUD)*, New York City, NY, USA, 27 June-2 July, 2015, Pages: 869-876.
  - **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*, Volume 10, Issue 5, September/October 2017, Pages: 797 - 810.

- Chapter 4 enhances the offloading decision algorithm in Chapter 3 by providing an online task offloading and scheduling algorithm that considers load balancing, context changes, and offloading benefits for the entire HMC environment. This chapter is derived from the following publication:

  - **Bowen Zhou**, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya, "An Online Algorithm for Task Offloading in Heterogeneous Mobile Clouds," *ACM Transactions on Internet Technology*, Volume 18 Issue 2, Article No. 23, March 2018.

- Chapter 5 demonstrates the group-based fault tolerant mechanism, GFT-mCloud, which is designed as a standalone module that can work with existing mobile cloud offloading solutions to adaptively provide different fault tolerant policies based on

task requirements and machine classifications. This chapter is derived from the following publication:

  - **Bowen Zhou** and Rajkumar Buyya, "A Group-Based Fault Tolerant Mechanism for Heterogeneous Mobile Clouds," *in Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2017)*, Melbourne, VIC, Australia, November 7 - 10, 2017.

- Chapter 6 proposes a reverse auction-based incentive mechanism to encourage mobile users to participate in the HMC offloading service with the guarantee of computation efficiency, truthfulness, and individual rationality for each participant. This chapter is derived from the following publication:

  - **Bowen Zhou**, Satish Srirama, and Rajkumar Buyya, "An Auction-Based Incentive Mechanism for Heterogeneous Mobile Clouds," *Journal of Parallel and Distributed Computing*, 2018 (Under review).

Chapter 2
Literature Review

Despite the rapid growth of hardware capacity and popularity in mobile devices, limited resources in battery and processing capacity still lack the ability to meet the increasing mobile users' demands. Both conventional techniques and emerging approaches are brought together to fill this gap between the user demand and mobile device's limited capabilities. In this chapter, we first discuss the background and key concepts in the heterogeneous and intermittent mobile cloud computing environment. Then, we provide a comprehensive taxonomy and survey of the existing techniques and frameworks for mobile cloud augmentation regarding both computation and storage. It focuses on the techniques aspect, following the idea of realizing a complete mobile cloud computing system, as to provide a guide on what available augmentation techniques can be adopted in mobile cloud computing systems as well as supporting mechanisms such as decision making and fault tolerance policies for realizing reliable mobile cloud services.

2.1 Introduction

RECENT years have seen the exponential development of smart mobile devices. They have gained enormous popularity among end users with more advanced mobile applications to enrich the user experience. However, compared to the growing demand for more enhanced user experience mobile applications from users, mobile devices are still in lack of adequate resources such as processing capability, storage and battery lifetime to provide such applications. The gap between the mobile hardware and users' demand will hinder the mobile devices from providing experience-rich mobile services.

Due to the slow development of the battery technology, mobile resource augmenta-

This chapter is derived from: **Bowen Zhou** and Rajkumar Buyya, "Augmentation Techniques for Mobile Cloud Computing: A Taxonomy, Survey, and Future Directions," *ACM Computing Surveys*, Volume 51, No. 1, Article No. 13, Pages: 1-38, ISSN 0360-0300, ACM Press, New York, USA, January 2018.

tion has been considered as a critical approach to tackling the gap. Augmentation techniques for resource-constrained machines have been investigated since two decades ago. In the early 1990s, the ideas of remote execution and inter-process communication were introduced to help leverage the computing resources of computer clusters and manage message passing traffic [85, 166, 212]. Remote execution aims to augment the computing capacity of resource-limited computers such as mobile devices and computers to run computation-intensive applications in a *client-server* mode. However, utilizing the distributed resources can bring many challenges, including parameter marshaling, client and service binding, and the semantics of remote invocation.

Later on, with the development of the Internet and remote execution, a new service paradigm called Service-Oriented Architecture (SOA) is introduced. Mobile web services (MWS) [174, 211] incorporate SOA with mobile devices to enable mobile device users to share existing software and services on other devices to augment its capability and conserve energy. Mobile devices in MWS can be either service clients or service providers.

Since remote execution and SOA rely on stable network connections, the performance of these systems can be unstable. Researchers study the possibility of utilizing the computing resources in the proximity (e.g., mobile devices nearby) with short-range wireless communications to provide an ad-hoc computing augmentation service. The earliest idea of the wireless ad-hoc network, "packet radio" network, was introduced by Defence Advanced Research Projects Agency (DARPA) in the early 1970s [106]. Due to the development of the mobile device and 802.11/WiFi, the mobile ad-hoc wireless network (MANET) was then developed in the mid-1990s to provide a self-configuring mobile device network without the requirement of network infrastructures for collaborative mobile computing. The disadvantage of MANET is that the dynamic nature of network topology caused by the mobility of devices requires high adaptability of the network functions. Moreover, this type of mobile device augmentation can only provide limited resources within the resource pool built with other mobile devices. Later, pervasive computing [198] was introduced for a ubiquitous computing paradigm that enables computing to migrate from any device to any other devices via any type of network as the user moves. This paradigm essentially makes it possible to merge the computing resources of desktops, servers and mobile devices to provide continuing services.

Recently, cloud computing emerged as a promising computing paradigm in both industries and academia. Cloud computing aims to leverage virtualization technologies to provide computing resources such as computation, storage, and networks as a utility. The advantages such as on-demand self-service, broad accessibility, elastic scalability and pay-as-you-go services make cloud computing a potential computing resource for mobile device augmentation. The computing paradigm that leverages cloud computing resources to enhance the performance of resource-constrained mobile devices is called mobile cloud computing. In some works, it is also described as cyber-foraging [131], which refers to the same computing paradigm as mobile cloud computing. We use the term "mobile cloud computing" throughout this chapter and the rest of the thesis. A significant amount of research has been proposed in this area regarding the approaches for mobile cloud augmentation, resource provisioning and management [52, 122, 209, 238, 242, 255]. These existing works focus on four main issues: the solutions of leveraging cloud resources, the efficiency of outsourcing the computation, the service availability and reliability, and the platforms for mobile cloud services.

As the hardware capabilities of mobile devices are developing rapidly nowadays, researchers have shown that the benefits of using the public cloud as augmentation resources for mobile cloud are decreasing [210], and the network conditions can be another performance bottleneck. On the other hand, utilizing a heterogeneous mobile cloud (HMC), namely a hybrid of mobile ad-hoc cloud, computers in proximity and public clouds, as augmentation resources can solve the issue. However, it also brings new challenges to the mobile cloud augmentation system.

Many surveys have been given on the mobile cloud computing [1, 137, 195, 203, 205]. Shiraz et al. [203] reviewed the existing distributed application processing frameworks for mobile devices to offload computational intensive applications to remote servers. It presented a detailed taxonomy and survey on the offloading frameworks including application partitioning, VM migration, partitioning granularity, migration objectives, etc. Abolfazli et al. [1] presented a survey on the existing mobile code offloading frameworks based on the different types of cloud-based resources, which include distant immobile clouds, proximate immobile computing entities, proximate mobile computing entities, and hybrid cloud. However, its main focus is on the cloud-based augmentation frame-

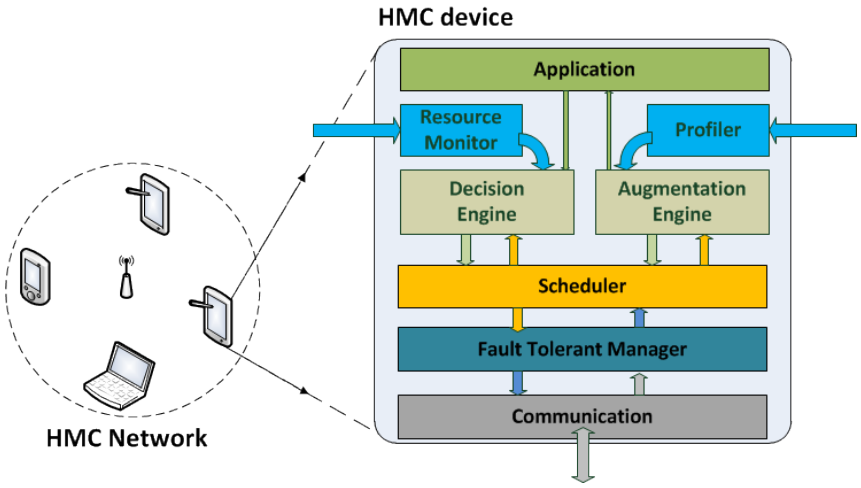


Figure 2.1: Typical modules of a heterogeneous mobile cloud system framework

works comparison, without a detailed classification and discussion of the techniques that can be used for implementing a mobile cloud system. Therefore, the interoperability of the techniques mentioned may not fit in the HMC environment. Moreover, the survey only provided a brief discussion on the supporting techniques of mobile cloud augmentation such as decision making, context monitoring, and fault tolerance. Sanaei et al. [195] took a look into the heterogeneity in mobile cloud computing. They presented a taxonomy of heterogeneity of mobile device and cloud on four levels for both, including hardware, platform, features, and APIs. The existing approaches for handling the heterogeneity in mobile clouds were discussed. However, their survey does not include the discussion on realizing computation and storage augmentation of the hybrid mobile cloud. Li et al. [137] focused on the survey of mobility-augmented cloud service provisioning and provided a taxonomy of cell network and cloud service provisioning mechanisms. Shuja et al. [205] focused on multimedia application oriented survey that discussed application virtualization, dynamic binary translation techniques, and native code offloading based mobile cloud frameworks. These two surveys provided discussion only on specific problems, and do not meet the technical demands of the new heterogeneous mobile cloud systems. Different from the existing surveys, in this chapter, we first introduce the concepts and issues of the HMC, and then we present a comprehensive taxonomy and survey of software techniques applied in the heterogeneous mobile cloud for both computation and storage augmentation.

The taxonomy covers the software augmentation related techniques for computation and storage and classifies with regards to implementation a heterogeneous mobile cloud system in practice. Figure 2.1 illustrates a typical set of modules for the heterogeneous mobile cloud system framework, where the techniques presented in the taxonomy are needed. The HMC network consists of several HMC system powered devices. On each HMC device, the *Augmentation Engine* implements the main technique for computation and storage augmentation. The application submits offloading task requests to the *Decision Engine*, which makes offloading decisions based on its decision logic and the context data observed from the *Resource Monitors* and *Profilers*. The *Scheduler* further schedules the tasks upon receiving offloading decisions from *Decision Engine* to other HMC devices for remote execution. The *Fault Tolerant Manager* module supports the task execution in the event of failures. The *Communication* module deals with all the data-related transmission. The taxonomy presented in this chapter is organized around modules of this system architecture.

In summary, the main contributions of this literature review are:

- It introduces the main concepts and most challenging issues in realizing mobile cloud augmentation systems.
- It classifies the state-of-the-art techniques and works in a taxonomy with two parts regarding mobile computing augmentation and mobile storage augmentation respectively. Each type of techniques and works is discussed with advantages and disadvantages, as well as the suitable conditions for adopting that type of technique.
- The taxonomy is organized following the idea of necessary modules needed for realizing a practical mobile cloud service, including outsourcing techniques, supportive decision making mechanisms, resource management, service reliability, etc.

The rest of the chapter is organized as follows. We explain the relevant terms and definitions, and the motivation of mobile augmentation in the mobile cloud computing in Section 2.2. Then we propose the taxonomy of computing and storage augmentation techniques and existing frameworks respectively in Section 2.3 and Section 2.4. Finally, we summarise the chapter in Section 2.5.

2.2 Overview

Different types of approaches have been applied for mobile computing and storage augmentation in mobile cloud computing. In this section, we explain the terms used in the previous works regarding the augmentation techniques. Furthermore, we discuss the motivations that have been targeted by the existing works.

2.2.1 Terms and Definitions

Due to the variety of techniques adopted in mobile cloud computing, we explain some of the important terms and conceptions that will be mentioned throughout this chapter and the rest of the thesis.

Cloud Computing

The cloud computing emerged in the recent years and gained popularity among both academia and industries. NIST gives the definition of cloud computing as "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models [159]."

A. Essential characteristics

On-demand self-service: Customers can quickly provision the services automatically according to their demand (e.g., process capacity, network, and storage).

Broad network access: Customers can access the cloud service over the Internet by using heterogeneous client platforms such as mobile devices, laptops, and workstations.

Resource pooling: The infrastructure providers pool many resources from data centers and provide resource renting for clients in a multi-tenant service manner, with multiple physical and virtual resources dynamically assigned upon consumer demand.

Highly scalable: The service provider can easily scale up and down with the resources to meet user's demand automatically or manually when needed.

B. Service models

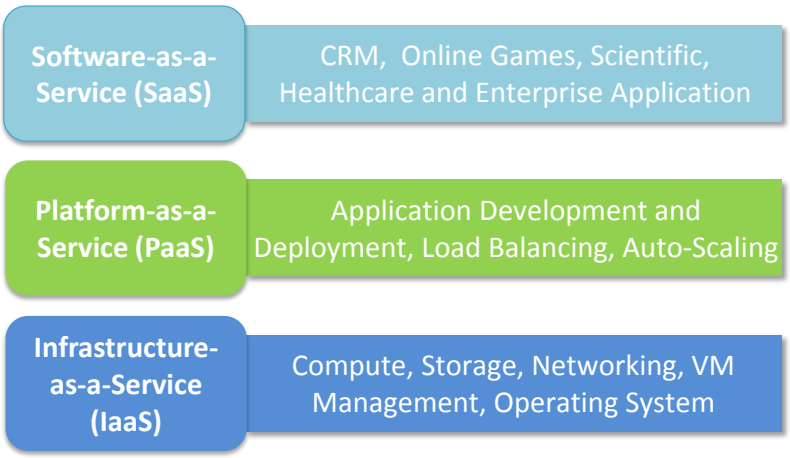


Figure 2.2: Cloud service models

Generally, there are three types of service models provided by cloud service providers. The models can be summarized in a service stack, which is shown in Figure 2.2. From the bottom to the top, IaaS provides virtualized data centre infrastructure resources such as compute, storage and networking. Examples include Amazon EC2, Microsoft Azure, and Google Compute Engine; PaaS provides platforms and tools for application and other development, testing and deployment. Examples include Majsoft Aneka and Apache Stratos; SaaS represents the largest part of cloud services. It provides web-based applications where the backend services are managed on the cloud. Examples of SaaS applications include Facebook, Gmail, and Office 365.

C. Deployment models

Based on the difference of the administrative domain of the cloud, it can be further referred as the public cloud for general public users, private cloud for private groups and companies, and community cloud. The cloud computing service is adopted by a significant amount of research work as a potential approach for resource augmentation in mobile cloud computing.

Mobile cloud computing

In the beginning, mobile cloud computing was defined as a restricted computing paradigm that considers interactions between only mobile devices and public cloud services. Later on, due to the instability of mobile devices as well as wireless networks, more types of

platforms and computing resources are introduced to mobile cloud computing to achieve seamless mobile cloud services. An overview of the heterogeneous mobile cloud service environment is shown in Figure 1.1. Based on the previous definitions and the new demands, we give the definition of mobile cloud computing as follows:

> Mobile cloud computing is a computing paradigm that enables the resource-constrained mobile devices to utilize heterogeneous computing resources (e.g., public clouds, private clouds, and MANETs) over multiple types of wireless networks (e.g., cellular network, WiFi, Bluetooth, and Femtocell) to provide mobile device users with a seamless, on-demand and scalable mobile service that has rich user experience.

The term "cloud" in mobile cloud computing refers to multiple types of computing resources. A brief classification of different clouds is as follows.

- **Infrastructure based cloud.** The infrastructure refers to public cloud services including IaaS, PaaS, and SaaS. The mobile device only outsources its computation and storage to the public cloud services via WiFi or cellular network.
- **Cloudlet based cloud.** *Cloudlet* refers to "trusted, resource-rich form-factor computer that is well-connected to the Internet and available for use by nearby mobile devices [200]". Cloudlets are deployed as middle layer servers between public cloud and mobile devices to reduce network latency.
- **Mobile device cloud.** It refers to a mobile wireless ad-hoc network (MANET), which consists of a set of mobile devices connected to each other via short-range wireless networks such as WiFi-direct and Bluetooth in dynamic topologies, with no support of networking infrastructure [51]. Mobile device cloud can further reduce the data transmission overhead and provide augmentation services in case the WLAN or public cloud services are unavailable.

2.2.2 Motivations

Mobile devices have already seen a significant improvement with more powerful and functional hardware such as multi-core processors, mobile graphic cards, and memory.

Nevertheless, realizing mobile cloud computing can bring numerous benefits to both mobile device users and cloud service providers. From the user's perspective, user experience on mobile applications and functions of mobile devices can be improved by introducing mobile cloud computing. For cloud service providers, mobile clouds enable a large user community to use their services. Additionally, providers can apply machine learning mechanisms on the service data of mobile device users to further provide more customized cloud services.

Computing capability

Despite hardware improvements seen in mobile devices in recent years, the slow processing speed and low RAM still hinder mobile devices from providing experience rich applications to end users [199]. Since mobile devices have been involved as part of people's daily activities such as social networking, office work, and gaming, they are expected to have PC approximate computing capacities. Mobile cloud computing provides the opportunities for developers to overcome this gap and enrich their applications by outsourcing computing-intensive tasks to cloud resources. Moreover, by migrating the computation to other computing resources, mobile devices are released to complete light tasks which eventually will improve the fluency of mobile devices.

Energy efficiency

As the hardware on mobile devices such as display, processor and sensors become more battery demanding, one of the most important issues smart mobile device users face nowadays is the battery efficiency. Since the currently battery technology cannot provide a significant performance improvement, an alternative solution needs to be considered, and mobile cloud augmentation can be a promising solution by bringing the cloud computing resources to the mobile device's proximity to enhance its battery lifetime.

Mobile storage

In the era of big data and mobile computing, many digital contents such as photos, videos, and large files have seen a drastic increase. Unlike the storage on PCs, mobile

devices provide only limited flash storage space despite the demands of large and elastic storage from mobile applications and services. The elastic cloud storage resources can be a solution to the lack of mobile storage. The characteristics of cloud storage such as elasticity, on-demand, and low cost make it a potential extension to mobile devices. However, due to unstable wireless connections and mobile device mobility, the offline usability of mobile cloud storage augmentation is one of the challenges. Moreover, the migration of sensitive personal data from mobile applications to cloud storage raises the concern of data security and privacy.

Seamless and collaborative mobile application

With the advent of smart mobile devices, mobile users expect continuous mobile application experiences by using mobile cloud computing services. However, the wireless mediums that carry out data migrations are the performance bottleneck for mobile cloud services due to its instability and accessibility. In the event of network infrastructure failures, public cloud services may not be available, and mobile applications running its computation intensive tasks will be interrupted. In such a case, an ad-hoc network of mobile devices as a collaborative resource pool can be an alternate solution. Another example is the mobile social network in proximity (MSNP) [31], which enables mobile device users to interact with other users nearby via peer-to-peer wireless communications and complete tasks collaboratively.

Many research works have been proposed with different techniques to tackle above-mentioned issues in heterogeneous mobile clouds, which include hardware augmentation technologies regarding battery [7], processor, etc. However, since our focus is on the development of heterogeneous mobile cloud systems for mobile cloud applications, this taxonomy covers the software augmentation technologies for HMC. A thematic taxonomy of the software mobile cloud augmentation techniques is depicted in Figure 2.3. It is divided into two main categories: computation augmentation and storage augmentation. For the computation augmentation, we present augmentation techniques related to task offloading for the *Augmentation Engine* in Figure 2.1, decision making techniques for devising offloading decisions for *Decision Engine*, with observed context monitoring data from *Context Monitor* and *Profiler*, and supporting techniques such as scheduling

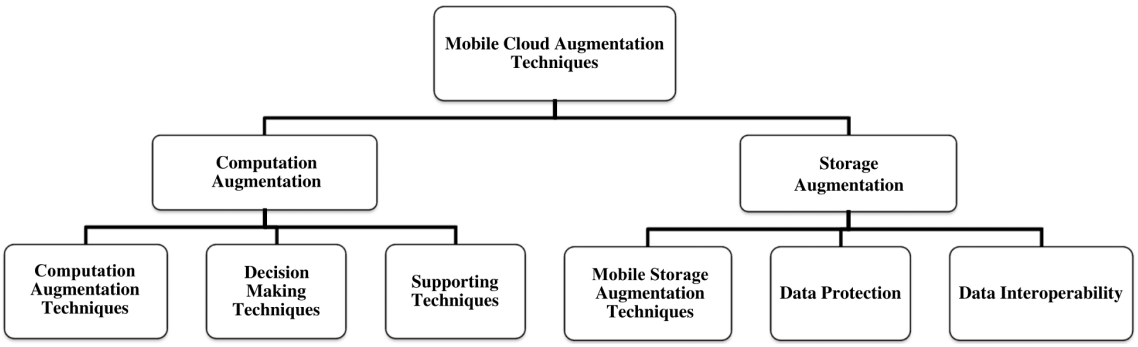


Figure 2.3: A thematic taxonomy of software techniques for mobile cloud augmentation

and load balancing, fault detection and recovery for the *Scheduler* and the *Fault tolerance* module. For the storage augmentation, we present the techniques related to storage offloading, data protection, and data interoperability. The detailed taxonomies of computation augmentation techniques and storage augmentation techniques are proposed in the following two sections respectively.

2.3 Taxonomy and Survey of Computing Capacity Augmentation Techniques

One benefit of mobile cloud computing paradigm is to bring the cloud resources to the device proximity to enhance the computing capability limited mobile devices. Multiple techniques can be leveraged to augment the computing capability of mobile devices. As shown in Figure 2.4, we will discuss the techniques in two aspects: the augmentation models and the augmentation architectures. Augmentation models, which include code offloading models and task delegation models, provide solutions of how the computation augmentation of mobile devices is realized by utilizing cloud resources. The augmentation architecture, which includes parallel execution and opportunistic mobile collaboration, describes the system architecture of the mobile cloud system for performing the computation augmentation. The two aspects work together to provide efficient mobile cloud augmentation services. A detailed discussion of these techniques is presented in the next four subsections.

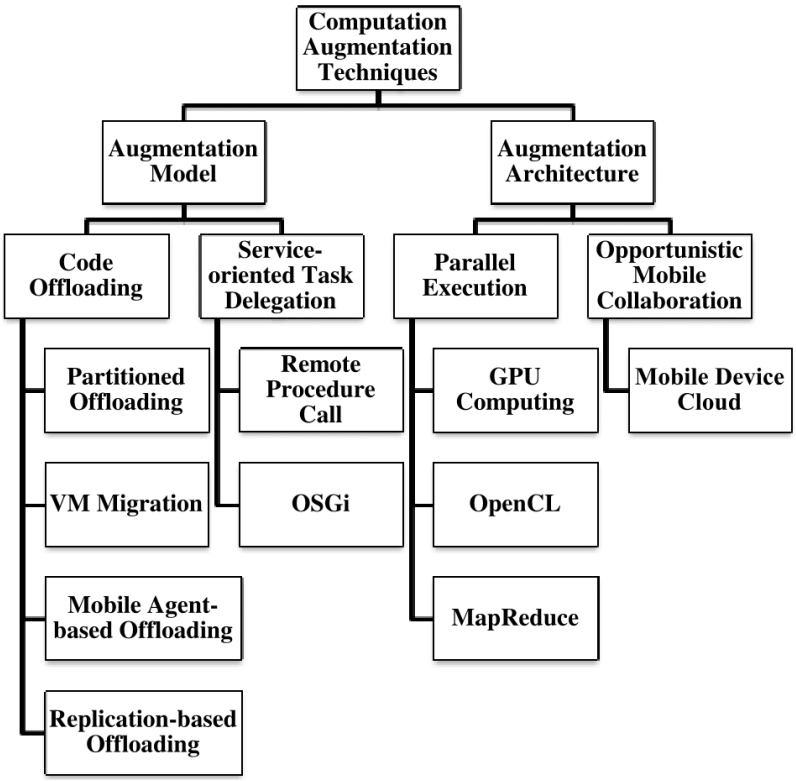


Figure 2.4: Taxonomy of computation augmentation techniques

2.3.1 Code Offloading

Offloading computation for remote execution is an idea that has been studied ever since the computer network was developed. The computation offloading technology aims at migrating the computation intensive code from resource-limited machines to remote computing resources to accelerate the running process of the computation and reduce the energy consumption on limited battery devices. Figure 2.5 shows a general concept of code offloading. In a general code offloading model, computation intensive codes of a mobile application are identified at first, and then it is evaluated by the decision making process based on the objective of the mobile cloud augmentation service (e.g., saving energy) to whether offload or not. Last, the code is offloaded to the remote computing resources by different types of available techniques. The code offloading technique assumes the entire application computation originally happens on mobile devices. It enables mobile devices to migrate part of the computation from resource-limited mobile devices to resource-rich computing machines, which makes it flexible in terms of out-

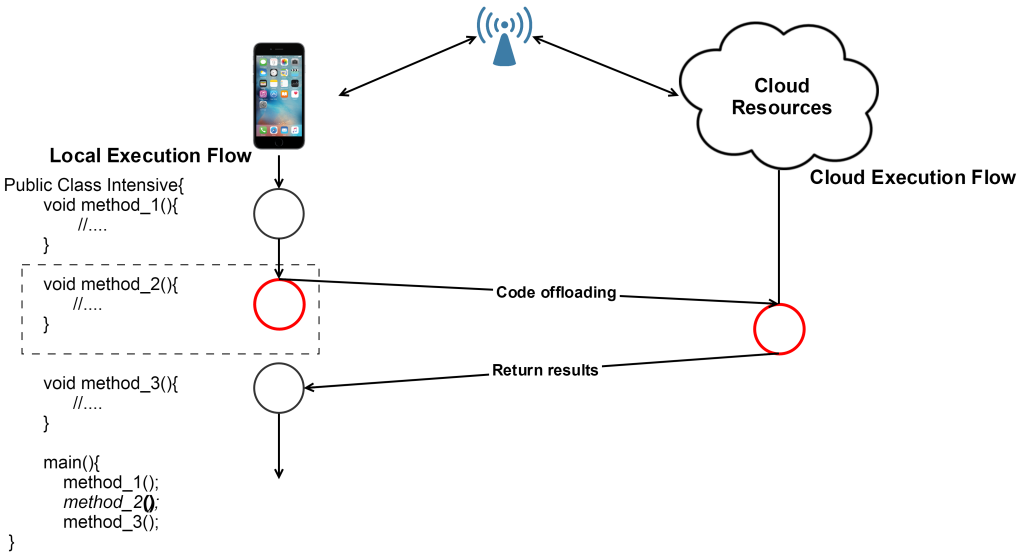


Figure 2.5: Code offloading concept

sourcing. However, the disadvantage of code offloading is that it requires developers to identify and partition the part of the computation to offload, which is a non-trivial task and may impose unnecessary overhead for mobile devices. Various code offloading approaches have been proposed and applied based on various programming models of the mobile applications. The approaches can be classified into four categories: partitioned offloading, VM migration, mobile agents based offloading, and replication based offloading.

Partitioned offloading

For partitioned offloading methods, only the computation intensive part of mobile applications is identified and offloaded to the remote servers. The conventional client-server computing model is introduced to perform the partitioned offloading for HMC augmentation. Resource-limited mobile devices are considered as thin clients, while the remote computing resources that execute the offloaded codes are considered as resource-rich servers. The application is partitioned either statically before the runtime or dynamically at runtime.

In the static partitioning, offloading partitions are decided and hard-programmed in the application by the developers. As a result, there is no overhead imposed by deciding

partitions when the application is running. However, static partitioning requires the comprehensive program running knowledge for developers which is a non-trivial task. Also, the static partitions make this offloading approach less adaptive to a dynamic computing environment like mobile clouds. Therefore, static partitioning is barely adopted in mobile cloud augmentation. To fix this issue, dynamic partitioning is introduced. Different from static partitioning, dynamic partitioning analyses the processes of an application at the runtime with the assistance of the offloading decision logic. Most commonly used dynamic partitioned offloading techniques are **Flow-based programming, .NET common language runtime programming, Java reflection**, and **distributed shared memory**.

Flow-based Programming (FBP). "FBP is a data flow programming paradigm that models applications as modulized processes connected with each other by pre-defined communication connections [161]". These modules can be reconnected with each other to develop various types of applications without having to be modified. Therefore, each module can be dynamically decided whether to offload. To apply FBP in mobile cloud computing, Hung et al. [97] ported JavaFBP onto Android system. The Android applications are coded with APIs provided by JavaFBP in modules. However, using FBP as code offloading approach requires the installation on both mobile devices and offloaded machines, which hinders the scalability of the applications. To solve this problem, another approach using **Java Reflection** is proposed by Kosta et al. [122].

Java Reflection is provided by Java that enables the program to inspect and manipulate the annotated classes, interfaces, fields and methods at runtime. Therefore, classes of an Android application can be decided dynamically at runtime whether to offload, and any machines with Java runtime can execute the offloaded partitions without requirements of additional installations. Kosta et al. [122] proposed *ThinkAir* for mobile cloud code offloading on the method level using Java reflection. The annotated methods are evaluated by offloading decision logics dynamically and offloaded to cloud VMs running Android clones. In this way, the offloading services can be easily scaled up and down on cloud VMs. However, by using Java reflection, ThinkAir is only able to offload one method at a time and may cause lock-in issues.

In order to overcome this lock-in problem, Gordon et al. designed another solution named *COMET* [80] for Android system using ***distributed shared memory***. It is built

on top of the Dalvik Virtual Machine [63] on Android that leverages the underlying Java memory model and VM synchronization to form a distributed shared memory (DSM) for the code migration between machines. The offloading is implemented by synchronizing the information of heap, memory stacks, register states, and synthetic classes between VMs on cloud and mobile devices.

Besides Java on Android, other mobile platforms also provide similar functions that can be used for code offloading. ***.NET Common Language Runtime (CLR)*** "provides a managed coding platform featuring cross-language integration and exception handling and a simplified model for component interaction [22]". Cuervo et al. proposed a framework called *MAUI* that adopts .NET CLR on Windows phones to enable code offloading. Similar to Java Reflection, methods annotated by CLR tags are evaluated by MAUI's decision making logic at runtime to decide whether to be offloaded. However, different from ThinkAir, MAUI requires users to develop a server version of the mobile application using CLR, which harms the application's scalability in mobile clouds. Moreover, MAUI lacks consideration of multi mobile device offloading environment, and same lock-in issue as Java reflection remains.

The above-mentioned four types of techniques are the major approaches proposed. Compared with the static code offloading, although dynamic code offloading is more flexible and able to adapt to the different execution environment, it still requires modifications of mobile applications at the application developing stage. This drawback makes it non-trivial and tightly coupled with the underlying programming model of the mobile applications. With the development of new smartphone operating systems, many works later presented VM migration techniques for computation offloading.

VM migration

Live VM migration approach [45] for code offloading is based on the idea of moving the computation dynamically among the machines in the distributed system without interrupting the ongoing execution. It is fairly suitable for computation augmentation in mobile clouds since most mobile applications, as well as cloud services, are running on virtual machines. VMs can be migrated either partially or completely based on the requirement of the offloading. Applications are not required to be installed on the server

side. However, the drawback of live VM migration is the overhead of transferring a VM image and its state via wireless networks can be costly and inefficient under the unstable network conditions. Hence, the focus of VM migration technique is improving energy efficiency considering the network conditions. Several previous works have proposed solutions based on live VM migration.

Satyanarayanan et al. proposed a VM migration based framework called Cloudlet[2] [200] to bring the computation from the mobile device to nearby form factor servers to overcome the long latency caused by transferring data to cloud via wireless networks. The term *Cloudlet* refers to "trusted, resource-rich computer or cluster of computers that is well-connected to the Internet and available for use by nearby mobile devices." *Dynamic VM synthesis* is developed to perform the transient partial VM migration service. The benefit of using Cloudlets is that it can be an alternative to the remote cloud resources to reduce the network bottleneck which often has significant effects on mobile cloud performances. However, the absence of monetary incentives of installing the Cloudlet-like machines is an issue to be further studied. Moreover, Cloudlets can only provide offloading service to stationary users. The continuous execution of the offloaded tasks would be an issue.

Chun et al. [43] took a different approach of using VM migration from Cloudlets. They proposed a code offloading framework *CloneCloud* that works with the application VM layer such as DalvikVM and Microsoft .NET. It deploys one or more clones of the mobile applications and data onto Cloud VMs and nearby servers and using process interception on the VM level to execute parts of the application on Cloud. The running states including data in the stack and heap are synchronized between the two processes on the mobile device and cloud VM. However, since the mobile clones have limitations on native data virtualizing, it does not have all the access to data on the cloud side, which narrows the range of functions to be offloaded. This is also a vital issue for all VM migration based code offloading approaches.

[2]Source code can be found on https://github.com/cmusatyalab

Mobile agent based offloading

In order to overcome the data access limit of VM migration for code offloading, Mobile agent based offloading is introduced to mobile cloud computing. A mobile agent is a movable software that can be transferred from one mobile device to a network and roam among the computing nodes in the network [40]. When an application using mobile agents needs to request a service from the remote server like the cloud, it collects the application execution information and passes to the execution environment of the agent for offloading and remote execution. There are several advantages of using mobile agents in mobile cloud computing. First, the mobile agent uses asynchronous communication to interact with remote servers in the network, which is suitable for mobile devices since the network connections of mobile devices are unstable. Moreover, the device can still perform other lightweight computation while waiting for the results from remote execution. Second, mobile agents are robust to server failures, since all the information including executing environment, the process states, services required, etc. are transferable. Some mobile agent supported frameworks have been proposed.

Angin and Bhargava [12] proposed JADE (Java Agent Development Environment)[3] for mobile code offloading. In the event of task offloading, the method being offloaded is transferred into an agent-based process by the feature *Behaviour* in JADE on either class level or method level. Then it consults the *cloud directory service* to select one of the cloud hosts for the remote execution. JADE provides the offloading code with high movability and running adaptively. However, JADE does not consider the QoS of its cloud resources such as load balancing, application multi-tenancy and wireless channel energy efficiency for multi-users.

The multi-tenancy issue of mobile agent based offloading is investigated by Liu et al. [145]. They considered the energy efficiency of the wireless channels with multi-user scenario and solve the problem by using Lyapunov optimization framework to minimize the number of transmission time slots based on the queue backlog and channel states on each mobile device. However, this optimization approach only focuses on data transmission without considering local and cloud task execution, which may have a significant impact on the efficiency of the proposed algorithm.

[3]Source code available at: http://jade.tilab.com/download/jade/

Replication based offloading

Either partitioned offloading, VM migration or mobile agent based offloading requires applications to precisely analyse the benefits of whether to run portions of the application on mobile devices or remote machines. As a result, it usually benefits applications with large computation and small data size. In addition, the above three techniques do not benefit network-intensive applications. In order to solve this problem, Gordon et al. [79] took a different approach by using replications. They proposed a framework called Tango for Android that attempted to reduce user-perceived application latency by switching execution and output display automatically between application and its replica on remote machines for the faster one. However, there are still some limitations of Tango. Only one replica can lead the execution instead of working simultaneously. This issue can affect the application running time as the leading replica may have no access to native resources such as sensors and user inputs. Also, the overhead of constantly switching is not presented in the work.

Discussion. The above-mentioned types of augmentation techniques adopted for code offloading aim to dynamically offload computation intensive code to cloud resources in order to conserve energy and speed up the execution. However, some code offloading approaches such as partitioned offloading require particular environment and comprehensive programming skills, which limits the usage on existing applications. Moreover, the lock-in issue of code offloading exists for approaches such as Java reflection and FBP. Also, the decision process of the offloading incurs additional overhead that will reduce the benefits of code offloading.

2.3.2 Service-oriented Task Delegation

Unlike the code offloading approach, service-oriented task delegation augments resource-limited mobile devices by utilizing existing services on remote servers with remote process invocation rather than migrating part of the application to the server for execution. OASIS has defined service-oriented architecture (SOA) as: "A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabili-

ties to produce desired effects consistent with measurable preconditions and expectations [150]."

A service is a self-contained unit that implements at least one reusable action such as searching a database or rendering a web page. The request to a service and its corresponding response are communicated via protocols such as XML and SOAP (Simple Object Access Protocol). The benefit of SOA for mobile cloud computing is that it provides a solution for multiple service providers and heterogeneous devices to incorporate their services into more comprehensive service combinations, where services are independent of any underlying vendors, products or techniques. The transparency of SOA-based services can reduce the additional issue and the execution lock-in issue of code offloading. SOA enables conventional applications to deliver their services to mobile devices via cloud resources with same user experiences on PCs, without the need of developing native mobile applications.

Web Service based Mobile Cloud

In the age of cloud computing, many legacy PC applications have been migrated to the cloud to provide the original functions in the form of web services. This SOA based service paradigm also enables mobile device users to use the legacy applications on their devices with the same PC-like user experiences via mobile cloud augmentation systems. The benefits of web service based mobile cloud systems are that there is no need to develop a native mobile application but only a thin client to relay the service requests to the cloud servers and render the returned results. In this way, the device is released to process other tasks and still able to obtain same, good quality of service. Many recent works have been proposed in the mobile cloud augmentation. Hani and Dichter [87] proposed a four-layer service-oriented architecture to provide an energy-efficient solution for web service based mobile cloud augmentation. The architecture ensures the security of the data handover and guarantees the quality-of-service. Rossi et al. [193] designed a cloud-based service architecture to provide geolocated emergency services on mobile devices. Mobile users report the real-time condition reports about the emergency via mobile devices to the backend services. Crowdsourcing techniques are applied to evaluate the circumstances on the cloud. Guerrero-Contreras et al. [86] studied the service availabil-

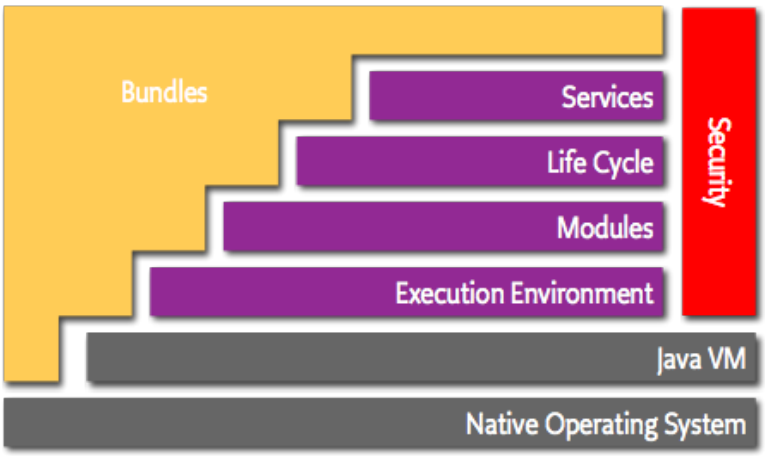


Figure 2.6: Layers of OSGi programming model [9]

ity in mobile cloud computing. They proposed a context-aware SOA based framework to improve the maintain service availability that can be downgraded by dynamic network changes in mobile cloud systems. The framework contains three services running on the cloud, including monitoring, context managing, and replica managing. As we can observe from these proposed works, one drawback of SOA based mobile cloud augmentation is service binding, which means mobile users can only choose certain provided services.

In the attempt to solve the operating binding issue, Flores et al. [71] proposed mobile cloud middleware *MCM*[4] for processing the intensive hybrid cloud services regardless of the underlying mobile operating systems. The middleware provides a set of cloud service APIs available to the mobile device users to build the applications in its own language. MCM also fosters the integration and orchestration of mobile tasks delegated with minimal data transfer. However, the cloud services provided in MCM are limited, and the ability of users to add additional cloud services is yet to be delivered.

OSGi (Open Service Gateway Initiative)

Another technique adopted in the SOA-based mobile cloud is OSGi[5] [9], which is a Java module management system enabling applications to dynamically load and unload ser-

[4]Source code available at: https://github.com/huberflores/MobileCloudMiddleware
[5]More details at https://www.osgi.org/

vice bundles at runtime. The difference of OSGi from the C based RPC is that OSGi is object-oriented. The bundles can be invoked by users with a service interface, which is registered in the OSGi service registry with an implementation of the interface. Figure 2.6 depicts the OSGi system stack. Bundles are OSGi components built by developers. Execution Environment defines the available methods and classes for the user's application. Modules define the way Bundles import methods and classes. Life Cycle provides APIs to install, start, stop, update, and uninstall bundles. Services dynamically bind the bundles with underlying Java objects. Services, life cycle, modules, and execution environment together define how the bundles work with underlying Java VMs.

Verbelen et al. [224] proposed an OSGi based middleware called *AIOLOS*[6] for mobile augmentation on Android. The bundle feature is ported to Android. The mobile application is developed with OSGi bundles, service interfaces and remote services that implement the bundle services. Junior et al. [107] proposed a mobile offloading system (MOSys) to enable seamless offloading when users move between wireless networks. It is implemented based on OSGi, utilizes software-defined networking and remote caching to reduce the response time and provide a seamless handover. A few other works [231, 244] have also presented OSGi based techniques to support the mobile cloud augmentation systems.

Discussion. SOA-based mobile cloud is fundamentally different from code offloading as there is no computation migration. Mobile devices in SOA work as thin clients that are only able to send service requests without any computation, while the main task computation is carried out in the form of services on the cloud. However, this limits the flexibility of mobile applications in terms of functions as it needs backend service to support.

2.3.3 Parallel Execution

More than one zettabyte of data is generated over the Internet nowadays [100]. The needs of processing big data surpass the capacity of mobile devices. The big data applications such as mobile forensics applications, data streaming applications and augmented reality applications have been challenging conventional mobile cloud computing augmentation

[6]The framework is available to download at: http://aiolos.intec.ugent.be/

approaches. Although code offloading and SOA-based task delegation can enhance mobile devices with more computing capability, the overhead of migrating a large amount of data to the remote server is getting higher and higher. As a result, the benefits of outsourcing the computation are counteracted.

On the other hand, utilizing parallel execution to distribute computation can significantly reduce the time overhead of handling big data applications. In parallel computing, the computation and related data are divided into subproblems with a chunk of data, which are then processed simultaneously on multiple resources. Regarding data-parallel applications, sets of either homogeneous or heterogeneous task are processed in parallel, and the results are merged to generate the final result. The benefits of using parallel execution for mobile cloud computing augmentation depend on the types of applications particularly requiring parallel data processing. The frameworks such as **GPU computing**, **OpenCL** and **MapReduce** have been adopted to cope with big data applications in mobile cloud augmentation.

GPU computing

The thousands of cores on GPU enable applications to offload compute-intensive portions to the GPU while the remainder of the applications still runs on the CPU. Soyata et al. [209] proposed a mobile cloud-based hybrid architecture (MOCHA) using GPU computing to enhance the performance of object recognition applications used on the battlefield. Typically, the battlefield handheld devices are connected via satellites that incur long latency. The Cloudlet in MOCHA is used to reduce the latency by letting mobile devices connect to Cloudlet in the vicinity via low latency links and then the Cloudlet processes either part or all the computation. GPUs are utilized to perform the massively parallel processing (e.g., object recognition), which provides massively parallel processing ability to applications by using CUDA[7], which is a C-like programming platform and programming models designed for GPU parallel programming. However, GPU computing is only suitable for mobile applications that involve matrix-based computation, e.g., image processing due to the structure of cores on GPU. In addition, only intensive computation on small size data instead of streaming applications is preferred so that the

[7]Download available at http://www.nvidia.com/object/cuda_home_new.html

speedup on execution would not be hindered by data transmission overhead. More importantly, GPU computing is only available with the certain brand of GPU, which causes vendor lock-in.

OpenCL

Different from GPU computing that only works with the particular type of applications, OpenCL [21] provides a more general parallel solution. It is an open-source platform for building parallel programs on cross-platforms that provide transparent C-compatible programming models for utilizing distributed processors regardless of the underlying architectures. Shih et al. [202] proposed an elastic computation framework for mobile clouds based on OpenCL frameworks. It federates computing and memory resources on wearable devices, mobile devices, nearby cloudlets and public cloud VMs as a shared resource pool for completing tasks on mobile devices. Eom et al. [64] studied the feasibility of applying HPC cluster architectures to the mobile cloud computing environment. They proposed an OpenCL-based offloading framework that migrates OpenCL workloads from the mobile grid to clouds. OpenCL provides more portability and compatibility comparing to GPU computing since it works on both CPU and GPU.

MapReduce

Another approach having been adopted as augmentation technique in the mobile cloud computing is MapReduce. "MapReduce is a programming model and an associated implementation for processing and generating large data sets with a distributed algorithm on a cluster [54]". Marinelli [154] developed a mobile-cloud computing infrastructure called Hyrax based on Hadoop. Hyrax enables smartphone applications to run in distribution both regarding data and computation by adapting the Hadoop onto mobile devices within the Hyrax system. However, NameNode and JobTracker instances are not realized in Hyrax. In Hyrax, these must be run on a traditional machine. Duo and Kalogeraki [58] implemented MapReduce paradigm on Nokia smartphones using Python. The master node including a scheduler, an HTTP server, and an application repository is implemented on a server to map the jobs to other mobile devices as worker

nodes. MapReduce is not widely used in mobile cloud computing since the processing capability of mobile devices is not enough for a complete MapReduce implementation.

Discussion. Different from the previous two augmentation techniques, parallel execution in mobile cloud aims to reduce the transmission overhead by breaking up big data into smaller chunks and process individually at distributed nodes at the same time in order to speed up the execution for resource-limited mobile devices. However, techniques like GPU computing have limitations on types of applications that can be run as well as vendor lock-in issues, which makes parallel execution restricted.

2.3.4 Opportunistic Mobile Collaboration

Mobile cloud augmentation techniques discussed above focus on utilizing remote computing resources via wireless networks like WiFi. However, most of these techniques rely on adequate wireless network bandwidth, which can be intermittent and cost-unfriendly. In addition, since mobile devices are usually on the move and may lose Internet connections in the area lacking signal strength, the communication between mobile devices and cloud services may be disrupted.

In recent years, the short-range wireless communication technologies on mobile devices such as Bluetooth, WiFi-direct, and Zigbee have been significantly improved regarding bandwidth and energy efficiency. With the development of both mobile device computing capacity and wireless networks, the mobile ad-hoc network (MANET) has become a potential mobile cloud infrastructure for augmentation. Many proposed works considered MANET as **mobile device cloud**, which is a network of mobile devices connected wirelessly in either centralized or distributed manner [218]. The mobile ad-hoc network provides a solution for mobile devices to dynamically utilize other peer devices in the vicinity to overcome the issue of unstable wireless network connections.

Many research works [153, 178, 187] have been carried out on applying MANET to enhance the mobile device performance. Penner et al. [178] proposed a collaborative computing platform, "Transient Clouds", to enable mobile devices in the vicinity to share their own mobile application services to other devices in the manner of ad-hoc networks. The prototype was developed on Android using the built-in WiFi-direct feature. The tasks are stored in the auto-generated .dex file and later distributed to other devices by

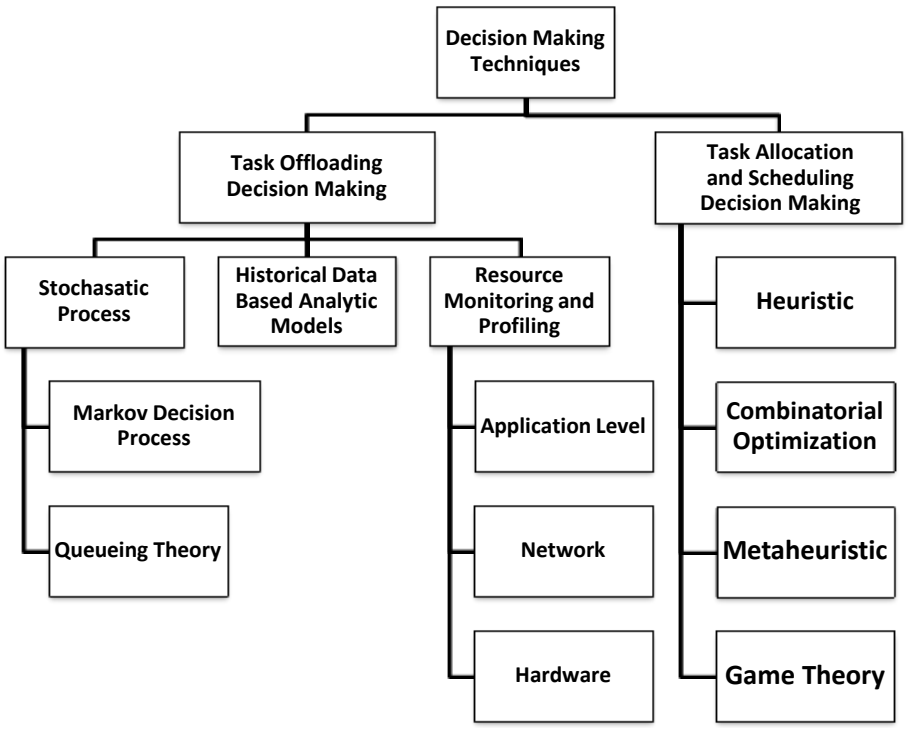


Figure 2.7: Taxonomy of decision making techniques

the decision engine. However, due to the limit of WiFi-direct, the ad-hoc network implemented is a one-hop network, which is not a complete implementation of the proposed framework and its feasibility needs further evaluation. Mao et al. [153] utilized mobile edge device computing to offload computation in order to reduce network latency and congestion, and proposed an online Lyapunov optimization-based dynamic offloading algorithm to minimize execution cost by adopting DVFS. The above-mentioned mobile device implementations all have only one type of wireless medium for communication, which leaves itself vulnerable to network disruptions, and also will not be scalable to devices with different wireless interfaces. Therefore, mobile device cloud that can utilize devices with multi-type wireless mediums needs to be investigated.

Discussion. The mobility and dynamic nature of MANET bring the reliability issue which needs the support of fault tolerant policies for a much stabler performance. The approaches addressing the fault tolerance issues in mobile cloud augmentation systems are discussed in Section 2.3.8. In addition, the performance of MANET can be affected by network congestion as well as synchronizing the up-to-date information of the mobile devices.

2.3.5 Task Offloading Decision Making

In order to provide efficient mobile task offloading services, decision making techniques are required to decide whether, when and where to offload. In this section and section 2.3.6, existing decision making techniques regarding task offloading, and task allocation and scheduling are discussed respectively. Figure 2.7 shows a detailed taxonomy of decision making techniques, including offloading decisions (whether to offloading), and task allocation and scheduling decisions (where to offload). For making offloading decisions, three main categories are discussed: stochastic process, analytic model, and resource monitoring and profiling. For making task scheduling decisions, the techniques are classified into four categories: heuristic, combinatorial optimization, metaheuristic, and game theory.

In this section, the task offloading decision making techniques are classified into three categories: 1)stochastic process, which abstracts systems with random variables and statistically devises offloading decisions as systems evolve into stable states with certain probabilities; 2)historical data based analytic models, which aims to abstract systems with parametric models based on historical data and devise real-time offloading decisions unlike the stochastic processes; 3) resource monitoring and profiling, which provides real-time monitoring and profiling of the system and running applications, and utilizes the real-time data to devise offloading decisions without analysing on parametric models.

For both parametric model-based techniques and real-time monitoring data based techniques, different factors of a mobile cloud system need to be captured in order to have valid analysis on the task offloading decisions. The factors can be classified as mobile application contents, contexts of the mobile cloud environment, characters of the hardware, and user preferences.

1) ***Mobile application contents***. The contents refer to characters of applications including application type, code granularity, data size and type, user interaction requirements as well as the computation intensity. These application contents have important impacts on the results of the augmentation. For instance, applications with computing intensive jobs are more suitable for cloud servers if outsourced. Data streaming applications such as video streaming and Optical Character Recognition applications require

short response delay while data analytic applications such as sentinel surveillance application in the medical field are more delay-tolerant. Hence, it is important to consider the differences in application contents when making augmentation decisions.

2) ***Context of mobile cloud environment***. The mobile cloud environment is a resource sharing environment that consists of mobile devices, remote servers, and various wireless communication mediums. For example, wireless communication is one of the key components when outsourcing the computation, and the bandwidth and congestion will affect the time taken for transferring data, which will hinder the augmentation performance. Moreover, the mobility of mobile devices adds dynamics to the environment as the available resources are changing rapidly. Thus the decision making schemes should be agile for dynamic context changes.

3) ***Hardware specifications***. The hardware refers to both mobile devices and remote servers, which have heterogeneous types of hardware such as CPU, memory, storage and communication modules. The challenge for decision making schemes is how to distribute the tasks among machines to either balance between time and energy saved by augmentation or fulfil user preferences based on the hardware as well as other factors discussed. Therefore, monitoring schemes are required to profile the hardware information to support the decision making schemes.

4) ***User preferences***. Last but not the least, user preferences which are priority objectives for decision making schemes should be taken into consideration. User preferences can be very different depending on users' circumstances. Some users want to use the augmentation service to save the battery of their mobile devices while other users wish to outsource the mobile tasks to get a shorter processing time regardless of how much battery it will consume.

Stochastic process

The stochastic process is a random process evolving with time [44]. It aims to abstract the evolution of a system that changes based on random variations and predicts possible outcomes weighted by probability. Some well-known stochastic processes include Markov processes, Poisson process, and queueing theory. In mobile cloud augmentation systems, a stochastic process is applied to model mobile applications as well as devices

statistically, and evaluate the execution conditions to help devise optimal augmentation schedules.

Chen et al. [35] proposed a semi-markovian decision process (SMDP) based method to solve the problem of optimal offloading tasks dispatching and transmission for mobile cloud augmentation systems. Its objective is to obtain an optimal trade-off between computation time and energy consumption. The semi-Markov decision process based optimization is developed to abstract mobile device status and decide the probability of tasks to offload and at each stable state, which DVFS level of CPU and data rate of the transmitter to be set. The SMDP is solved with linear programming. One drawback is that many mobile devices have no access for users to change DVFS and wireless interface status.

Instead of focusing on a single device, Terefe et al. [216] proposed an energy-efficient multi-site offloading policy for mobile clouds using Markov decision process (MDP). MDP is used to formulate application partitioning and scheduling to multi-clouds problem as a delay-constrained, least-cost shortest path problem with the goal to minimize energy consumption on wireless channels. Similarly, Wang et al. [229] applied Markov model in dynamic scheduling for mobile cloud health telemonitoring. As the mobile cloud infrastructure in health monitoring faces ever-changing clinical priorities and personal demands, the MDP based dynamic offloading scheduling algorithm proposed solves a joint optimization of processing latency, energy efficiency, and diagnostic accuracy. The stochastic process such as MDP can provide optimal solutions to multi-objective problems in reality. Nonetheless, the processing overhead of solving the problem and the sensitivity of the solution to new changes may pose a burden on the resource-limited mobile devices.

Historical Data Based Analytic models

Unlike stochastic process, this approach uses parametric models to abstract multiple attributes (parameters) of the mobile cloud augmentation environment, analyses the benefits of augmentation in terms of time and energy consumption based on historical data of the task execution. The attributes range from CPU speed of mobile devices as well as remote servers, network conditions (e.g., bandwidth, congestion, and latency), to the

load of remote resources. However, the disadvantage of analytic models is that the types of attributes used in the models have to be consistent with the objective of the models. Too many attributes without proper assumptions and too few attributes without accurate abstraction may both downgrade the performance of the models. Many previous works applied this method to evaluate benefits of the mobile cloud computing augmentation.

Ali et al. [6] focused on modeling the power consumption of mobile devices to enhance the energy efficiency of mobile cloud augmentation systems. They presented detailed energy consumption models for CPU, display, wireless communication interfaces and memory on the mobile devices. For example, the energy model for CPU is based on the CPU frequency and utilization:

$$Power_{cpu} = \beta^{freq} \times U + \beta^{freq}_{base}, \tag{2.1}$$

where β^{freq} and β^{freq}_{base} denote frequency dependent coefficients when CPU is in the busy and idle state respectively. U is the CPU utilization. All the coefficients in the energy models are derived from a linear regression on the measurement results of smartphones. However, this regression approach is device dependent and may not be accurate when applied to other mobile devices.

Rahimi et al. [184] took into consideration device mobility when making augmentation decisions by modeling mobile applications as a *location-time workflows*. The mobile device is assumed to move in a partition of a 2D area. The mobility sensitive workflow models are derived by integrating the locations (i.e., coordinates) of the mobile device and time duration of the device on that location into the workflow models. It is represented as follows:

$$W(u_k)^L_T \triangleq (w(u_k)^{l_1}_{t_1}, w(u_k)^{l_2}_{t_2}, \cdots, w(u_k)^{l_n}_{t_n}), \tag{2.2}$$

where u_k is the k^{th} mobile user and $w(u_k)^{l_n}_{t_n}$ is the user workflow in location l_n for time t_n. Such models can assist mobile cloud augmentation services to meet QoS requirements considering mobile device movements. However, the accuracy of the proposed models needs to be further studied and improved since the mobile user trajectory are obtained from some conventional mobility models such as Random Point Walk and Manhattan models [27].

Chen [37] proposed a set of analytic models for processors and wireless network interfaces in terms of both computation and communication. For the computation models, the author proposed execution time models and energy consumption models which are both related to the computation size of tasks. The models are as follows:

$$T_n^l = \frac{D_n}{F_n^l},\ E_n^l = \nu_n D_n, \tag{2.3}$$

where D_n is the computation amount, ν_n is the coefficient of the energy consuming rate per CPU cycle, and F_n^l is the computation capability. Similarly, for the communication, the models are proposed based on the size of input data B_n, the network speed R_n, and wireless network power consumption rate P_n of user n.

Resource monitoring and profiling

In order to overcome the issues of context adaptability in analytic model approach, resource monitoring and profiling are adopted in mobile cloud computing. This approach involves the implementation of multiple profilers and monitors on mobile devices to constantly monitor the context changes and the behaviour of the mobile device. Then the obtained data is put through the decision logic to evaluate the execution benefits and makes augmentation decisions based on the objective and user preferences. Many works have been proposed using this approach. We will discuss a few representative ones.

Benkhelifa et al. [17] proposed a genetic algorithm based resource augmentation for mobile device cloud to minimize energy consumption. A social cloud resource profiling and negotiating system are implemented, including a logger module which focuses on profiling the energy usage of applications on mobile devices under different circumstances such as using different wireless interfaces, executing tandemly with other applications, and executing at the different time of the day. However, the process of a large amount of gathered data against the storage limited mobile device is not presented.

On the other hand, history-based resource profiling with compressed data storage model can solve the above issue. Kaya et al. [112] implemented a set of profilers to monitor the application usage such as running time and CPU usage on mobile devices and cloud VMs. The history-based profiles are then constructed in call graph model that

axes redundant information to reduce the storage. In case the application has never been executed before, the profiling mode would be invoked first and run several times with all possible use cases. However, this process puts unnecessary extra time for the application users, which would affect user experiences.

ThinkAir [122] implements a set of profilers on Android that observe a more comprehensive range of system parameters to assist its *Execution Controller* for augmentation, including hardware profilers, application profilers, and network profilers. All the profilers are implemented using the system APIs provided by Android.

Discussion. In this section, we discussed three major types of offloading decision making techniques. The stochastic process discussed above provides a statistical approach of devising offloading decisions based on the states of the system and the transition probabilities obtained from different distributions. The accuracy of statistical distributions abstracting the mobile cloud systems can have a significant impact on the offloading decision. The analytic model approach has a similar component of system abstracts as stochastic processes, but instead, it devises offloading decisions based on historical execution data of the system rather than statistic distributions. Unlike these two approaches that are unagile to the context change of system environment, the resource monitoring and profiling approach make offloading decisions based on real-time observations of task executions in mobile cloud systems. The main difference between the three approaches is the execution data that they make offloading decisions. Note that these three approaches can be used as combinations to provide more accurate results.

2.3.6 Task Allocation and Scheduling Decision Making

In a heterogeneous mobile cloud environment, each mobile device can outsource tasks via augmentation techniques to other resources using wireless networks. After the execution monitoring on the context parameters is completed, mobile tasks submitted from mobile devices need to be allocated to execute either in local or offload to other computing resources. Moreover, these tasks also need to be scheduled among the shared resources pool to achieve the objective of the system, such as optimal task completion time or minimum energy consumption. Therefore, task allocation and scheduling algorithms are required to distribute mobile tasks among the shared resources. Although

there have been many task scheduling algorithms developed for distributed computing environments such as grid and cloud computing, the unique characteristics of heterogeneous mobile clouds such as intermittent wireless networks, unstable devices, and human-related behaviours make it difficult for the conventional scheduling algorithms to be applied. Hence, new algorithms and mechanisms are needed. Many task scheduling algorithms have been adapted to solve task allocation and scheduling problems in heterogeneous mobile clouds. They can be classified into four categories: **heuristics, combinatorial optimization, metaheuristics, and game theory**, based on their approach to solving the allocation and scheduling problem, as shown in Figure 2.7.

Heuristics are search algorithms that rank and select the solution subjects at each step of the searching process based on the most current information. They are often considered as a faster, less complex type of algorithms for solving optimization problems [175]. This is suitable for executing the algorithms on mobile devices that require lightweight processes. Greedy algorithms, listing heuristics, Min-Min algorithm are among many of the mostly used scheduling algorithms proposed for mobile cloud augmentation. Li et al. [132] proposed six online and batch scheduling based heuristics for scheduling independent tasks onto mobile nodes in mobile device cloud to reduce energy consumption. Trneberg et al. [222] proposed an iterative local search algorithm to find near-optimal solutions for application placement on mobile cloud resources. However, in exchange for the processing efficiency, heuristics are only able to provide local optimal solutions as they have no knowledge of the future events.

On the contrary, combinational optimization can devise global optimal solutions for the task scheduling optimization problems in mobile cloud augmentation. It searches the entire space of the feasible solution candidates and finds the best one as the optimal solutions. The computation efficiency of this method can be exhaustive if the search space is significantly large. Therefore, it usually deploys on a more powerful tier of computing resources in the heterogeneous mobile clouds. Methods such as (integer) linear programming, dynamic programming, and A* search algorithm have been applied in the mobile cloud augmentation. Gai et al. [73] presented a dynamic programming based algorithm to minimize the energy consumption of wireless communication in mobile cloud systems. Yang et al. [243] solved a joint optimization of task placement and load balancing

with linear programming, and further proposed a greedy heuristic with low complexity. As we can observe, heuristics are possible alternatives when combinational optimization is infeasible to solve in real time. Another alternative is approximation algorithm, which aims to reduce the original problem to a similar, conventional problem, and is solved with approximation algorithms to the reduced problem. Some recent works have proposed solutions with this method [108, 197, 240].

Another alternative is metaheuristic, which is a high-level algorithm to select and guide a heuristic to generate near-optimal solutions. They often refers to the nature-inspired algorithms such as genetic algorithms, ant colony algorithms, and simulated annealing. Metaheuristics usually adapt randomization and stochastic process in each of its evolutionary searches to reduce the search space and generate the solutions in a reasonable amount of time. Many recent works have been proposed to solve task scheduling problem in heterogeneous mobile cloud augmentation using metaheuristics such as genetic algorithms [17, 72, 82], swarm optimization [81, 180], and ant colony optimization [186, 232].

Game theory based methods can also be leveraged to solve optimization problems, especially for mobile cloud augmentation where mobile users can be considered as players in task outsourcing games. It can abstract the heterogeneous mobile cloud in a more realistic approach in which the mobile device users can apply some strategies to maximize their interest. The game theoretic scheduling algorithm aims to find the optimal task augmentation schedules that can achieve a Nash equilibrium so that no strategy change from one player can change the results of optimal solutions. A few recent works have presented the game theory based algorithms for task offloading and scheduling in mobile cloud systems [37, 38, 215].

Discussion

The task allocation and scheduling algorithms proposed in mobile cloud augmentation systems aim to provide optimal mobile task execution solutions to achieve the system objectives, such as optimal execution time and minimized mobile device energy consumption. As we discussed, the performance of these algorithms depends on the types of mobile applications, user preferences, SLAs (service level agreement), and the context

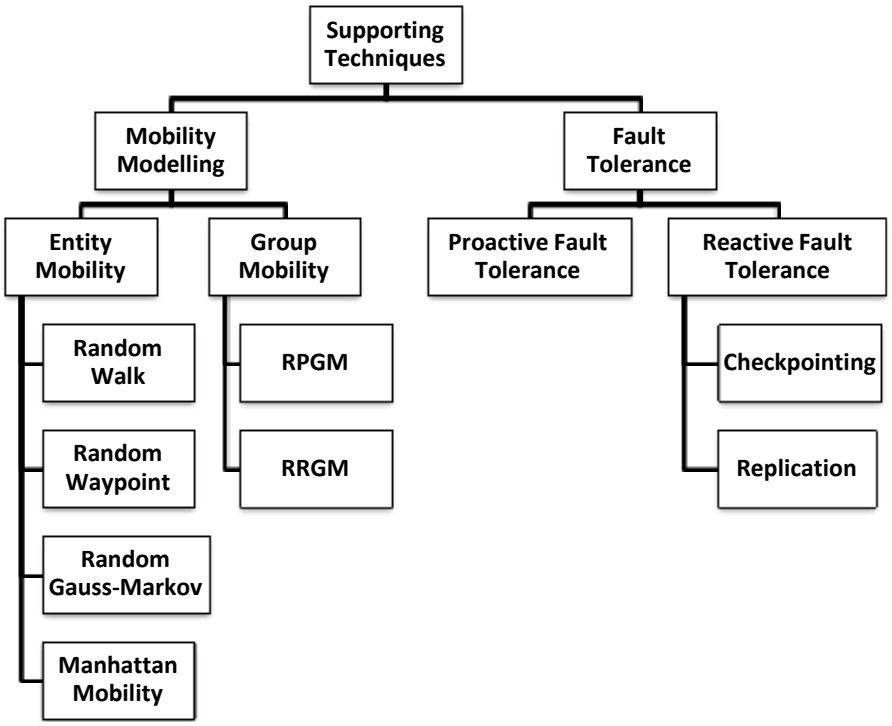


Figure 2.8: Taxonomy of supporting techniques for computation augmentation

of mobile cloud augmentation systems. Therefore, the algorithms need to be designed with the consideration of all the factors discussed.

2.3.7 Mobility Model for Mobile Device Cloud

In the following two sections, we discuss the mobility impact of mobile devices and fault tolerance techniques caused by the device's mobility. These are the supporting techniques for the heterogeneous mobile cloud augmentation. A detailed taxonomy of the supporting techniques is presented in Figure 2.8. The mobility modeling is important for mobile cloud computing, especially for mobile device cloud which consists of an ad-hoc network of mobile devices. The decision making logics in a mobile cloud augmentation system can utilize the information obtained from the mobility models to estimate the availability and reliability of the mobile devices in the network and further make augmentation decisions. There are two types of mobility models commonly used: trace-based models and synthetic models. Trace-based models refer to the models built on real-world moving traces, which contains accurate information of the mobility behav-

iors. However, the real-world traces require long period observation and a large number of participants. Therefore it is only useful case by case. On the other hand, synthetic models try to capture the mobility patterns without the assist of traces. The synthetic mobility models are categorized as entity mobility models and group mobility models.

Entity mobility models

Entity models aim to mimic the moving patterns of a single mobile node, without the interaction with other nodes in the proximity. We present four most commonly used entity models: Random Walk model, Random Waypoint model, random Gauss-Markov model, and Manhattan mobility model. A few available simulators such as ns2, ns3 can be used to simulate the node mobility.

The Random Walk model was firstly introduced by Karl Person [176] in 1905. It describes a walk path that consists of a series of random steps on a single or multiple dimension space. In the mobile cloud scenario, it is usually a 2-D space. At each stepping point, the mobile node chooses a random direction from [0, 2π] and a random speed from a range set by the model, so that the patterns can be limited to a certain area. The Random Waypoint model is very similar to Random Walk model. The only difference is that Random Waypoint model adds a randomly chosen pause time to every stepping point. These two mobility models have been widely applied in mobile cloud computing [134, 155, 214]. However, from the description, we can observe that Random Way model is a stateless random process that does not remember that information of previous steps. This property makes Random Walk generate unrealistic walking patterns since there may be sudden stops or sharp turns.

In order to solve the sudden stops and turns issue, the random Gauss-Markov model (RGM) can be applied in mobility modelling for mobile cloud augmentation. RGM moves mobile node in time intervals. At each time interval, the next location d_{next} and speed s_{next} are calculated based on its current location d_{pre} and speed s_{pre}:

$$s_{next} = \alpha s_{pre} + (1-\alpha)\bar{s} + \sqrt{(1-\alpha^2)s_{ran}} \tag{2.4}$$

$$d_{next} = \alpha d_{pre} + (1-\alpha)\bar{d} + \sqrt{(1-\alpha^2)d_{ran}} \tag{2.5}$$

where α is the tuning parameter to vary the randomness. $\bar{s}, \bar{d}$ represent the mean values, and s_{ran}, d_{ran} are two random variables from a Gaussian distribution. RGM avoids the sudden change issue by letting past states influence the future states. A few works have RGM implemented [13, 160, 181].

For some special mobile devices like devices on vehicles, nodes move on an urban grid which includes only horizontal and vertical movements. Manhattan mobility model captures such movement patterns. In this model, each mobile node at each intersection is allowed to choose to go straight with 0.5 probability or to turn left or right with 0.25 probability. This model is implemented in [103, 184].

Group mobility models

Different from entity mobility models, group mobility models consider the influence of mobile device between each other that may affect the movement. In mobile cloud systems, especially mobile device cloud, nodes tend to move together such as a tourist group or a group of soldiers. Two most commonly applied models are Reference Point Group Mobility (RPGM) model and Reference Region Group Mobility (RRGM) model.

RPGM [91] is one of the most used group-based mobility models. Each group will have a central node as the leading node which follows an entity mobility model and sets the speed and direction for the entire group. The other mobile nodes in the network will be paired with a reference point that follows the leader point's movement in same direction and speed. When the reference points move to the new location, its associated points will move to a random location within a circle of radius R around the reference point. A few RPGM based mobility models have been proposed for mobile cloud augmentation in [11, 94]. However, the RPGM model should avoid using Random Walk based models that could generate sudden stops.

RRGM [167] further extends the RPGM to use a real-time determined sequence of regions to lead the group to some destination. The reference region is determined dynamically by user-defined node density and the size of the group. Members linked to the reference region will move to a random location in the region and then follows random waypoint model to wait until all members arrive in the region. The group will eventually move to the target destination following the reference regions. The advantage of RRGM

is that it can mimic the real scenario where a large group can be divided into sub-groups and merge back together after some movements.

2.3.8 Fault Tolerance

As a computing environment with heterogeneous mobile devices, remote servers, and wireless network interfaces, maintaining a continuous service to avoid and recover from service interruptions caused by failures is one of the difficult challenges. Existing fault tolerance strategies proposed in distributed computing can be applied to mobile cloud augmentation environment with adaptations to its unique challenges such as device mobility and loss of wireless connections.

Mobile devices play the significant part of mobile cloud systems as it is not only the consumer of the systems but also can be resource providers (i.e., mobile device cloud). As resource providers, mobile devices can enter and leave the system environment unpredictably in case it moves out of the wireless network range or the network connection is disrupted. Therefore, the faults considered in the context of mobile clouds are related to the mobility and availability of the mobile devices and remote servers as well as the stability of wireless network conditions. There are two main types of fault tolerance strategies: proactive fault tolerance and reactive fault tolerance.

Proactive fault tolerance

Proactive fault tolerance aims to prevent system faults by monitoring system through preventative measures to predict potential faults and prevent them from taking place. When a node failure is indicated, the fault tolerance mechanism preemptively migrates parts of the mobile application from the nodes about to fail. The basic form of proactive fault tolerance applies feedback-loop control over the distributed nodes in the system, i.e., mobile devices and cloud servers. The feedback loop consists of continuous node monitoring and reallocation of application partitions. However, proactive fault tolerance policies predict only before the task is dispatched, with no further action even if failure happens during task execution.

Hummel and Jelleschitz [96] proposed a proactive and reactive combined fault tol-

erance mechanism for the ad-hoc mobile grid. The proactive fault tolerance policy uses two mechanisms: *redundant execution* and *super-peer access*. Super peers refer to nodes that are constantly available and able to perform critical tasks. The unstable nodes use super-peer access to perform the execution and usually apply redundant execution on more than one node. Park et al. [171] proposed a Markov process-based approach to analyse and predict resource states to improve the system's resistance to fault problems related to the mobility of mobile devices. Ravi and Peddoju [187] presented a handoff strategy for the offload mobile tasks that proactively monitor the execution conditions of the mobile cloud augmentation system. The proactive failure evaluation adopts fuzzy logic based multi-criteria decision making algorithm. The handoff strategy is decided by monitoring energy consumption produced by using services from other resources and the remaining available time of mobile devices.

Reactive fault tolerance

On the other hand, reactive fault tolerance policies aim to reduce the effect of system faults that already happened. Mostd applied reactive fault tolerant mechanisms are checkpointing, and replication.

Checkpointing allows applications to restart at the most recent checkpoint when a failure is detected. Sonara [36] is a platform that aims to provide continuous mobile cloud augmentation service. Sonara's execution engine enables a fault-tolerant distributed runtime that performs checkpointing based partial rollback recovery scheme. It adopted Chandy and Lamport's snapshot protocol [30] that carries out a global checkpointing throughout the system periodically.

Replication strategy is another well-known fault tolerance mechanism. In order to keep the service operate continuously after system faults occurred, multiple replicas of the task are distributed and run on different resources until the task is complete. However, replication will bring the challenge of adding redundancy into the system as well as synchronization problem. Choi et al. [41] proposed a fault tolerance scheduling algorithm for a content addressable network (CAN) based mobile cloud augmentation system. This is done by using cloud service replication. When the cloud server receives a request for cloud service, the server returns with two or more proper resources from

other mobile devices that can meet the QoS and operate the service on all the resources.

A summary of the existing implemented frameworks in the mobile cloud computation augmentation techniques is presented in Table 2.1.

2.4 Taxonomy and Survey of Mobile Storage Augmentation Techniques

Mobile devices are equipped with limited storage that cannot match mobile application's growing needs of larger data storage consumption. On the other hand, public cloud also provides the Data-as-a-Service (DaaS). DaaS enables users to offload a large amount of data to cloud storage services that mobile device storage cannot provide. For example, Amazon web service provides Simple Storage Service for object storage and charges based on the usage. Therefore, data storage augmentation by storing mobile data on cloud servers is of interest for mobile cloud augmentation systems. The sensitive data transmission such as location coordinates and healthcare information via wireless communication mediums and data storage on other computing resources bring security challenges into mobile cloud computing. We present a taxonomy of mobile storage augmentation in Figure 2.9.

2.4.1 Mobile Storage Offloading

Despite that mobile devices have been improved regarding hardware, the lack of substantial storage and computing capacity are still hindering mobile devices from better user experiences. Cloud has become a primary resource for enhancing mobile devices in terms of computation as well as storage.

Table 2.1: Comparison of mobile cloud computation augmentation frameworks

Framework	Cloud Resource Type	Code Offloading	SOA Task Delegation	Parallel Execution	Opportunistic Collaboration	Context-aware Execution Monitoring	Task Allocation & Scheduling	Mobility Management	Fault Tolerance
MobileFBP	Nearby servers	Flow-based programming	—	—	—	Resource profiling	Single user, heuristic	—	—
MAUI	Nearby servers	.Net common language runtime	—	—	—	Resource profiling	Single user, minimum makespan and energy, ILP	—	Checkpoint
ThinkAir	Public clouds	Java reflection	—	—	—	Resource profiling	Single user, heuristic	—	Memory error detection
COMET	Public clouds	Distributed shared memory	—	—	—	Thread execution profiling	Single user, heuristic	—	Checkpoint
Cloudlet	Nearby servers	Live VM migration	—	—	—	—	Single user, heuristic	—	—
Clonecloud	Public clouds	Live VM migration	—	—	—	Analytic models	Single user, minimum makespan and energy, ILP	—	—
JADE	Public clouds	Mobile agent(Java)	—	—	—	Analytic models	Single user, heuristic	—	—
Scavenger	Nearby servers	Mobile agent(Python)	—	—	—	Resource profiling	Multi users, heuristic	—	—
Spectra	Nearby servers	—	Remote procedure call(RPC)	—	—	—	Single user, heuristic	—	Proactive prediction on usage demand
Cuckoo	Nearby servers	—	Android interface defined language	—	—	Not required	Single user, heuristic	—	—
SAMI	Public clouds	—	SOA-based middleware	—	—	Not required	Multi users, heuristic	—	—
MCM	Public clouds	—	SOA-based middleware	—	—	Not required	Multi users, heuristic	—	—
AIOLOS	Public clouds	—	OSGi	—	—	Not required	Multi users, heuristic	—	Checkpoint
AlfredO	Mobile device cloud	—	OSGi	—	—	Not required	Multi users, heuristic	—	Redundant execution
MOCHA	Mobile device cloud and public clouds	—	—	GPU computing with CUDA	—	Analytic models	Multi users, heuristic	—	—
Hyrax	Mobile device cloud	—	—	MapReduce	—	—	Single users, heuristic	—	Redundant execution
[202]	Mobile device cloud and public cloud	—	—	OpenCL	—	Analytic models	Multi users, heuristic	—	Imprecise computation model
[64]	HPC clusters	—	—	OpenCL	—	Resource profiling and discovery	Single user, heuristic	—	Monitoring error code implemented in each method
mCloud	Mobile device cloud, Cloudlet, public cloud	Live VM migration	—	—	Ad-hoc network	Resource profiling and discovery	Single user, heuristic	—	Heartbeat, checkpoint
Transient Clouds	Mobile device cloud	—	—	—	Ad-hoc network	Resource profiling and discovery	Multi users, heuristic	—	—
Handoff scheme	Mobile device cloud and public cloud	—	—	—	Ad-hoc network	Resource profiling and discovery	Multi users, heuristic	—	Fuzzy multi-criteria decision making based handover strategy
Follow-Me Cloud	Public clouds	—	SOA-based middleware	—	—	Resource profiling	Single user, minimum makespan, MDP	Random Walk Model	—
[181]	Mobile device cloud	—	—	—	Ad-hoc network, edge computing	—	Multi-user, MDP	Gauss-Markov model	—
MobiCloud	Mobile device cloud	—	SOA-based middleware	—	Ad-hoc network	Resource profiling	—	RPGM	—

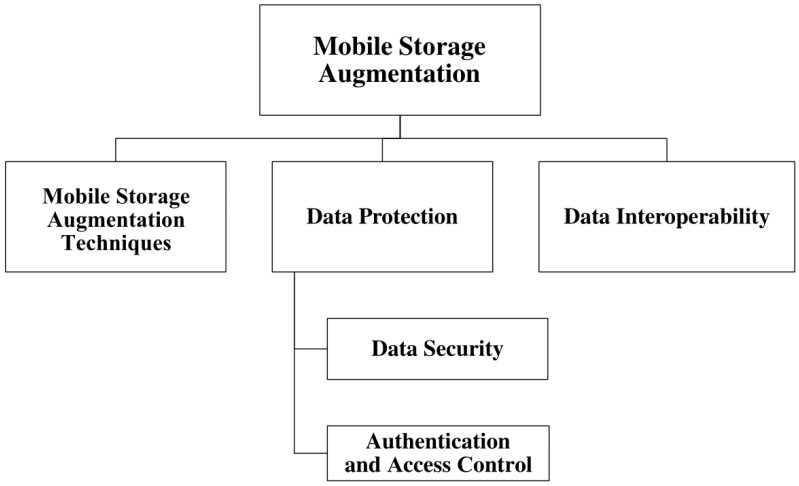


Figure 2.9: Taxonomy of mobile storage augmentation

Many approaches and techniques have been adopted in mobile cloud storage augmentation to solve the problems such as storage augmentation, data distribution, data access, and data synchronization.

Most commonly used mobile storage augmentation solutions involve public cloud storage services. In order to extend mobile storage, the solutions propose middleware between cloud services and mobile devices to provide data offloading and data management functions. Zheng et al. [253] presented a novel cloud storage augmentation framework called *SmartBox*. It introduces a concept called "shadow storage" services to extend the storage on mobile devices. The shadow storage service will automatically back up the data stored on mobile devices to the cloud storage when the device is connected to SmartBox, and share data between different mobile devices. Hung et al. [97] proposed a mobile storage augmentation system based on Trusted Architecture for Securely Shared Service (TAS3) [118]. It is a secure network for user-centric storage by using a rule-based policy framework to let service users store and process their private data in distributed applications on both mobile devices and cloud servers. However, mobile storage augmentation using public cloud services heavily relies on wireless networks, which makes it a disadvantage in case there are wireless network outages.

In order to avoid such wireless network bottleneck, some approaches applied mobile device cloud as the storage augmentation resources. Phoenix [170] is a protocol designed to make opportunistic use of mobile devices in the MANET to provide a short-term stor-

age service to clients in the proximity of each other. The distributed and asynchronous protocol breaks content into blocks and stores copies of blocks on multiple mobile devices so that it can ensure some degree of storage redundancy despite hardware failures, device mobility, and wireless communication failures. Additionally, Phoenix implements an advertisement model for maintaining and managing the blocks among the distributed mobile device storage. Similarly, Chen et al. [34] proposed a fault-tolerant data storage management algorithm for mobile device cloud that only contains mobile devices. The algorithm introduces a "k-out-of-n" strategy, which is a well-studied reliability control strategy [48]. The strategy distributes multiple copies among the number of n mobile devices and ensures that the system operates correctly as long task copies are available.

Different from using mobile device cloud to solve the network bottleneck, Cui et al. [53] investigated the wireless network bandwidth and data sync traffic directly. They proposed a system called *QuickSync* to improve the synchronization inefficiency problem in mobile cloud storage augmentation services.

2.4.2 Data Protection

For most mobile applications such as healthcare applications and OCR applications that get benefits from mobile cloud augmentation services, the data in transmission often contains sensitive and private information, and therefore needs to be protected. We discuss three major related issues, namely data security, accessibility, and authentication.

Data security

There are three important attributes of data security, which are confidentiality, integrity, and availability.

Confidentiality guarantees that the sensitive data is only exposed to users with proper authority. One of the commonly used approaches for confidentiality is data encryption. By encrypting the data with private information (e.g., public/private key pair), it can only the authenticated users have access to the data. Examples include SSL/TLS and HTTPS which are communication protocols for secure information exchange.

Data integrity maintains the consistency and accuracy of the data. The most widely

applied methods for protecting data integrity include data hashing technologies such as MD5 and SHA that capture the signature of the original data with certain hashing methods and compare it with the received data.

Data availability refers to the property that authorized entities can exclusively access the information on demand. A common solution for improving data availability is doing regular offline data backups, which is used to restore the data and services when the original data is damaged or under attack like the DDoS attack. Many works have been proposed for the mobile storage augmentation to ensure the above-mentioned three attributes of data security. Common topics include encryption, access control, authentication, data synchronization, and privacy.

Zhou and Huang [256] present a new Privacy Preserving Cipher Policy Attribute-Based Encryption (PP-CP-ABE) to ensure the confidentiality of the data. The encryption algorithm is based on CP-ABE [18], which is used to simplify the process of key pairs generating and access control. However, CP-ABE requires intensive computation that is not suitable for mobile devices. PP-CP-ABE provides a solution to this problem by outsourcing encryption and decryption operations to cloud with privacy preservation by performing the last step of decryption at the decryptors.

Yuan and Yu [248] investigated the data integrity checking techniques for cloud data sharing with multi-user data modification. They proposed integrity checking scheme with constant computational cost by using polynomial-based authentication tags which allow aggregation of tags of different data segments. Therefore, the scheme can tag files in batches from different users on the cloud and ensure a constant performance with scalability. Wang et al. [226] proposed a similar approach but without the need of entire data file for integrity checking. This is enabled by utilizing re-signatures for file blocks on a public verifier.

Li et al. [135] adapted Chase and Chow's attribute-based encryption (ABE) scheme [32] into mobile cloud computing with substantial communication and computation overhead reduction to provide a low complexity multi-authority ABE scheme (MA-ABE) mobile devices sharing their storage. The overhead reduction is made by introducing a cloud server based semi-trusted-authority, where the computation of keys for encryption and decryption are taken places.

Different from the above approaches, Khan et al. [115] consider the limited resource on mobile devices when adopting data encryption for mobile cloud computing. They proposed an increment-based proxy re-encryption scheme that improves file modification operations. A trusted entity was introduced so that the re-encryption services are offloaded on it instead of performing on the mobile device itself.

Authentication and access control

In mobile cloud storage augmentation systems, mobile users usually share files among multiple computing resources as well as other mobile users. To support the data protection requirements, mechanisms for access control and authentication need to be provided to ensure only the verified user groups have access to certain files.

Zhou and Huang [256] proposed a set of access control schemes called Attribute Based Data Storage (ABDS) for energy-efficient and secure storage augmentation services in mobile cloud computing as well. The ABDS access control is managed by an access policy tree that consists of leaf nodes and branch nodes. The leaf nodes represent parameters that carry the information of the access request, and each branch node is a logical gate, such as "AND" and "OR". The mobile users requesting the MSA services will be determined by each access policy tree defined for different user groups.

Lomotey and Deters et al. [148] designed a framework called *ALILI* to solve the group file sharing problems in mobile cloud systems. ALILI aims at ensuring data synchronization and user authentication among the mobile device users as well as cloud service platforms. The authentication is based on OAuth 2.0 mechanism to enable the user with secure tokens and basic information retrieved from social media credentials servers. It also makes the authentication process easier to map the user shared files in a shared storage.

Wang et al. [228] proposed a key distribution mechanism for mobile devices in Internet of Things, considering real-time data collection and monitoring. The proposed mechanism secures real-time key distribution in batch for the parallel mobile services while keeping the communication cost consistent with the number of mobile clients.

2.4.3 Data Interoperability

The heterogeneity of mobile cloud augmentation systems brings the problem of data interoperability. Since mobile devices may have different operating systems and hardware settings, the application data, and APIs for communicating with each other may vary in data formats. Issues such as data interoperability between various service APIs on cloud and mobile devices, data portability among different types of data warehouse facilities, and data migration from mobile devices to cloud services or across different cloud service vendors need to be studied.

One of the solutions is applying standardized service frameworks and message exchanging techniques such as SOA, REST, XML and JSON to mobile augmentation systems. Abolfazi et al. [2] proposed a SOA-based mobile device cloud called *MOMCC*. Since SOA defines standard service APIs for mobile devices and cloud services and services apply SOAP to exchange messages, mobile applications only need a small amount of modification to provide services across heterogeneous platforms.

Mobile social networks (MSN) are social networking where mobile device users having common interests connect and interact with each other through their mobile devices. In MSNs, the heterogeneity of software platforms on mobile devices and intrinsic user data and content raise the need to develop uniformed mobile application platform. Toninelli et al. [219] proposed a middleware called Yarta for mobile social systems. To achieve this, a representative model based on the Resource Description Framework [8] is presented. The mobile social applications developed with the representative model can share and reuse their respective data interoperably.

Doukas et al. [59] presented a mobile healthcare application framework for Android OS that provides trusted healthcare information online storage, retrieval, and update using cloud services. The data management of healthcare data in mobile cloud augmentation systems involves problems such as data privacy and interoperability. They presented a data management system called *HealthCloud* that implements a series of REST APIs for utilizing the storage services on the cloud. To ensure the data security, all the transferred data are compressed and sent via SSL enabled links.

[8] https://www.w3.org/TR/rdf-primer/

2.5 Summary

We presented a comprehensive taxonomy and survey on the augmentation techniques applied in mobile clouds. Firstly, the terms and definitions used throughout the survey are explained, including cloud computing and mobile cloud computing concepts. Secondly, the existing mobile cloud augmentation techniques for both computation and storage are discussed in two detailed taxonomies. The taxonomy of computing capacity augmentation discussed the approaches and techniques that have been applied in mobile cloud augmentation to merge the hybrid cloud resources into a shared resource pool for mobile devices to provide reliable and energy-efficient computation outsourcing mobile-cloud-as-a-service. The advantages and disadvantages of each type of technique are compared, and the challenges for existing technologies to adapt the new hybrid mobile cloud computing environment are identified. Regarding mobile storage augmentation, the taxonomy discussed the data-oriented architecture for storing data on distant clouds as well as mobile device cloud. Moreover, it covered the most critical issues in mobile cloud augmentation systems, namely data protection and data interoperability.

From the literature, we have identified that most of the existing technologies focus on building solutions for a onefold mobile-to-cloud architecture, which lacks the adaptability to the more powerful and efficient heterogeneous mobile cloud environment. Therefore, in this thesis, we aim to design a mobile cloud offloading enabling architecture for the heterogeneous mobile cloud environment (Chapter 3), with context-aware task offloading algorithms (Chapter 3, and 4) and supporting algorithms including a fault tolerant mechanism (Chapter 5) and an incentive mechanism (Chapter 6) to provide the seamless mobile-cloud-as-a-service.

In the next chapter, we present the proposed context-aware task offloading algorithm for heterogeneous mobile cloud offloading service as well as the offloading enabling technique and a designed mobile task offloading service architecture.

Chapter 3

A Context-aware Task Offloading Decision Algorithm and Enabling Framework

As a dynamic network of mobile devices and cloud resources connected via wireless mediums, the context of the mobile cloud offloading service changes constantly. To realize the benefits of heterogeneous mobile cloud offloading services in terms of performance enhancement and energy efficiency, an efficient offloading enabling technique and corresponding service framework are required to provide a cross-platform approach for different mobile devices and cloud platforms. In this chapter, we propose a context-aware offloading decision algorithm aiming to provide code offloading decisions at runtime on selecting the type of computing resources to offload a task as well as the wireless medium for data transmission. Also, we propose an offloading framework called mCloud to enable the task offloading algorithm proposed. The framework is implemented on Android platform, and is used to evaluate the performance of the proposed offloading decision algorithm.

3.1 Introduction

MOBILE cloud computing (MCC) provides services by bringing abundant resources in cloud computing [25] to the proximity of mobile devices to improve the mobile applications performance and conserve the battery lifetime. One of the tech-

This chapter is derived from:

- **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 Proceedings of IEEE 8th International Conference on Cloud Computing (CLOUD)*, New York City, NY, USA, 27 June-2 July, 2015, Pages: 869-876.
- **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*, Volume 10, Issue 5, September/October 2017, Pages: 797 - 810.

.

niques adopted in mobile cloud computing is code offloading [52]. It identifies the computation intensive code of a mobile program, and offloads the task to a cloud service via wireless networks. In the concept of code offloading, cloud resources used for offloading have many different types. First, the most common resource is public cloud services, such as Amazon, Google, and Microsoft Azure, which provide pay-as-you-go services over the Internet. Second, a nearby server named cloudlet [200] can be considered as a cloud resource with fast network connection as well as powerful processors. Cloudlet serves as a middle layer between mobile devices and public cloud services to reduce the network delay and accelerates the computing. Last but not the least, a local mobile device ad-hoc network forming a device cloud [213] is another potential cloud resource, especially when there is no access to the Internet.

Code offloading in MCC has been comprehensively studied during the past few years [43, 80, 122, 124, 154, 252]. These works mainly focus on the code partitioning and offloading techniques, assuming a stable network connection and sufficient bandwidth. However, the assumption is rather unrealistic. As in reality, the context of a mobile device, e.g. network conditions and locations, changes continuously as the device moves throughout the day. For example, the network connection can be unavailable or the signal strength is low. Since a mobile device usually has multiple wireless mediums, such as WiFi, cellular networks and Bluetooth, and each connection performs differently in terms of speed and energy consumption, the strategy of utilizing wireless interfaces can significantly impact the performance of the mobile cloud system as well as the user experience. Moreover, as we described above, there are multiple options of cloud resources that can be selected for code offloading under different conditions. As an example, in case the Internet connection is inaccessible, a group of mobile device users can still configure a code offloading service by setting up a wireless mobile ad-hoc network. Alternatively, a mobile device user can also connect to a nearby cloudlet to outsource the computation intensive mobile tasks when it is infeasible to use mobile data. The heterogeneity of MCC has not been rigorously studied in the literature as previous works only target the public cloud service.

To tackle the issues mentioned above and improve the service performance in mobile cloud computing, in this chapter, we propose a context-aware decision making algorithm based on Multi-Criteria Decision Making (MCDM) methods that takes the advantages

of the changing context of a mobile device and multiple cloud resources to provide an adaptive mobile code offloading service, and design a system framework called mCloud to provide mobile cloud task offloading services. The objective of mCloud is to devise offloading decisions under the context of the mobile device and cloud resources to provide better performance and less battery consumption.

The remainder of this chapter is organized as follows. We first present an insight of mCloud architecture in Section 3.2. Then we introduce the system models and propose the context-aware offloading algorithm in Section 3.3. The approaches for system implementation are introduced in Section 3.4, followed by a discussion on the system evaluation and experiment results in Section 3.5. Finally in Section 3.6, we summarize the chapter.

3.2 System Architecture

In this section, we give an insight of mCloud's architecture to address the issue of context aware code offloading in a heterogeneous mobile cloud environment. We first describe an overview of the MCC environment that mCloud fits in. Then, the design of main components is presented in details.

3.2.1 System Overview

Figure 1.1 illustrates the overview of the proposed system for the heterogeneous mobile cloud environment. The device requesting the offloading service is regarded as a *client*. The proposed system leverages three types of mobile cloud resources, namely cloud, cloudlets, and mobile device clouds.

First, cloud provides Infrastructure-as-a-Service with scalable computation and storage, which can be connected from mobile clients via WiFi or cellular network. It has the ability to host tasks requiring high computation and communication.

Second, a cloudlet is a local "datacenter in a box", which resembles a cluster of multi-core processors and a high-bandwidth WLAN connection with considerable low power consumption [200]. A task with limited delay-resistance is well-suited in cloudlets.

Third, a mobile device cloud that is formed by a group of ad-hoc mobile devices in

the client's proximity via short-range wireless network technologies, e.g. Bluetooth and WiFi Direct. Due to mobility of the devices, the mobile device cloud can be unstable. To ensure the stability of the mobile ad-hoc cloud in the mCloud framework, we adopt a one-hop topology for the mobile ad-hoc network.

3.2.2 Framework Components

The main components of the mCloud framework belong to two parts: client part and cloud server part. As depicted in Figure 3.1, there are five main components on the client side: *Context Monitor, Decision Module, Task Manager, Communication Manager* and *Failure Recovery*. On the cloud server side, main components include *Communication Handler, Task Manager* and corresponding mobile cloud infrastructures.

Context Monitor

The *Context Monitor* offers the context-awareness for the system by profiling multiple context parameters at runtime, and assists the *Decision Engine* when needed. Since the context of the mobile device has significant effect on the decision making accuracy, the system provides three relevant profilers: a program profiler, a device profiler, and a network monitor. However, profiling incurs additional runtime overhead and extra energy consumption. To avoid the high overhead, the profilers adopt the on-demand monitoring strategy, which only fetches context data when the offloadable methods are invoked.

a) Program Profiler: The program profiler tracks the execution of a program on the method level. The attributes being monitored include:

- the overall instructions executed,
- the execution time,
- the memory allocated,
- the number of calls of this method,
- the type of mobile cloud infrastructure resource for the execution (e.g. local, cloud, cloudlet),

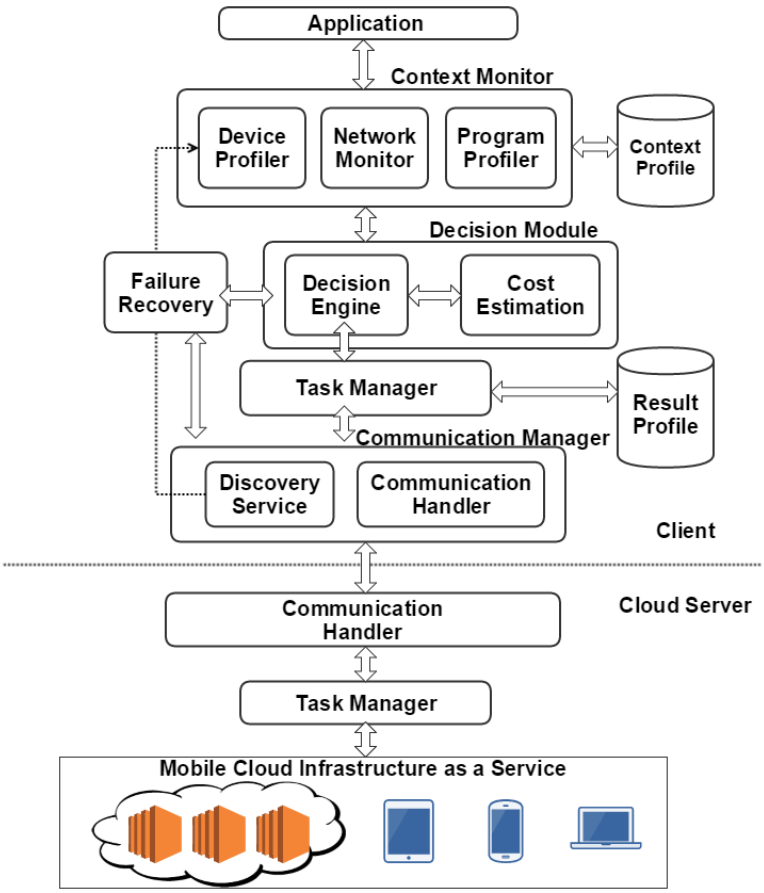


Figure 3.1: Main components of the framework

- and the data size of inputs.

The profile is updated at every invocation, and stored in the *Context Profile* database. Later the program profile is passed to *Cost Estimation* module to estimate the execution cost (i.e. running time, energy consumption) for decision making. The details of cost models are discussed in Section 3.3.2.

b) Device Profiler: The hardware profiles collected by the profiler represent the operating conditions of the mobile device being monitored. Same as the program profile, the device profile is fed into *Decision Engine* when needed to assist the cost estimation. The profile includes:

- the average CPU frequency,

- the average CPU usage,
- the maximum CPU frequency,
- and battery level.

c) Network Monitor: The network monitor collects the network information of the mobile device asynchronously at runtime so that it can record any change in the context. The profile is passed to cost estimation models when needed. The following network conditions are monitored:

- cell connection state and its bandwidth,
- WiFi connection state and its bandwidth,
- Bluetooth state,
- the congestion level of the connection (RTT) to VMs on the cloud,
- and the signal strength of cell and WiFi connection.

Decision Module

This component has the responsibility to decide whether and how to offload the mobile task, and dispatch the task to the appropriate mobile cloud infrastructure (i.e. cloud, cloudlet, or MANET) based on the current context. It consists of two main modules: *Cost Estimation* and *Decision Engine*. Based on the context profiles, *Cost Estimation* provides a set of cost estimation models that calculate the execution time of each offloadable task running on three types of mobile cloud infrastructure respectively, with the corresponding energy consumption on the client. *Decision Engine* then applies the cost estimations to the proposed context-aware decision making algorithm to provide the offloading decisions. We give a detailed discussion on the cost models and decision making algorithm in Section 3.3.

Task Manager

This component works as a middle layer between *Decision Module* and *Communication Manager*. It receives the decision, i.e. method name, offloading location, and network

interface, from the upper layer. Then the Manager collects the related information, such as the method inputs, libraries for running the offloaded task, and network address of the offloading location. Last, it persists them into a format called *Task Specs* and passes the information to *Communication Manager*. When receiving the task results, *Task Manager* stores them in the device database.

Communication Manager

The communication manager on both client and server side handles connections between client mobile device and the remote execution in either mobile ad-hoc cloud or remote cloud VMs. It consists of a mobile cloud infrastructure discovery service and a communication handler. Once the decision engine generates the offloading decision, the communication manager takes over the task and executes based on the offloading decisions. Moreover, the *Communication Manager* corporates with *Failure Recovery* service to detect failures and start the recovery process.

a) Discovery Service: This module is responsible for discovering the mobile cloud infrastructure resources, i.e. the available mobile devices in the MANET, cloudlets, and cloud VMs. The service updates the information of the detected resources, such as network congestion level, IP address, computation capacity, etc., and stores the information in *Device Profiler*. Particularly, for the mobile ad-hoc cloud, the *Discovery Service* detects the available mobile devices in the proximity, forms an mobile ad-hoc cloud, and maintains the network at runtime. Nevertheless, the discovery via network interfaces can potentially incur additional overhead and energy consumption. To avoid the high detection overhead, the *Discovery Service* applies the *periodic detection* strategy, which asynchronously searches for the available devices in certain intervals periodically. We present the detailed implementation of the *Discovery Service* in Section 3.4.3.

b) Communication Handler: The Handler operates on both client and mobile cloud infrastructures to handle the communications and data transfer generated in between, which include mobile cloud infrastructure detection, offloading code, state synchronization between client and servers, failure detection, etc. In particular, when offloading a mobile task, the *Communication Handler* on the client serializes the code, related input, and the information of libraries needed to execute the code into the offloading package.

Then it dispatches the package to the address provided by *Task Manager*. On the server side, the *Handler* unpacks the package and starts to synchronize the missing libraries described in the package with the client device. Once all the states are synchronised, it passes the deserialized code to the mobile cloud infrastructure for execution, and returns the result to client device upon completion.

Failure Recovery

It works with *Decision Module* and *Communication Manager* to detect the failures and recover the system. The details regarding the types of failures handled by Failure Recovery service, and the detection algorithm are presented in Section 3.3.4.

3.3 Cost Estimation Model and Algorithm

We first introduce the system model and the problem formulation. Then we present the details of our cost estimation model and the context-aware offloading algorithm.

3.3.1 System Model and Problem Formulation

The system considers a heterogeneous mobile cloud environment, consisting of a mobile ad-hoc cloud, the nearby cloudlets and remote public cloud. There is a set of mobile device users that run applications seeking opportunities to offload tasks to the mobile cloud infrastructure.

Task modelling

Different mobile applications have different QoS requirements. For example, face detection application requires short processing time while anti-virus applications are usually delay-tolerant. In this model, we model the tasks being offloaded as independent and can be partitioned into subtasks for parallel execution. Thus, let t denote the task generated by the application,

$$t = \langle s, w \rangle \tag{3.1}$$

s is the file size of the task, and w denotes the number of instructions of task t to execute[2]. All the symbols used in the models are listed in Table 3.1.

Mobile ad-hoc cloud modelling

We apply a one-hop mobile ad-hoc network (MANET) in mCloud to improve the network stability. This is because using multi-hop MANET can cause considerable delay and increase the possibility of node failures. Moreover, when the number of hops is more than two, cloud VMs are the most preferred as they have less communication delay in comparison to MANET [69]. Hence, we only consider a one-hop MANET in mCloud.

Given the one-hop network topology, we assume that the node movement within the signal range does not affect the topology of the MANET and the channel data rate remains the same. Let $M = \{ m_1, m_2, ..., m_n \}$ be the set of available mobile devices in the mobile ad-hoc cloud. μ_n denotes the CPU speed of node m_n. τ_n denotes the link congestion level between node m_n and the client device. Then the mobile ad-hoc cloud can be modelled as

$$m_n = \langle \mu_n, \tau_n \rangle, \forall m_n \in M \tag{3.2}$$

Cloud and cloudlet modelling

As described in Section 3.2, the client connects to cloudlet via WiFi, and to cloud via WiFi or cellular network. On both cloud and cloudlet, we deploy single-core VMs as the offloading solution. VMs within a cloudlet or cloud are considered homogeneous, while they are heterogeneous across clouds and cloudlets in terms of computation capacity and network delay. Then we model VMs on cloud and cloudlet as follows.

$$v_i = \langle \mu_i, r_i, \theta_i \rangle \tag{3.3}$$

where μ_i is the CPU speed, r_i is the network delay from the client to VM, and θ_i is the average CPU usage.

The heterogeneity of VMs between cloudlet and cloud reflects on the different μ, r values in the model.

[2]Our system considers the Android Dalvik bytecode instructions

Table 3.1: Notations of cost models

Symbol	Description
t	the mobile task being considered to offload
s	the data size of the offloadable task t that need to be transferred during offloading, in byte
w	the number of instructions for task t to complete
M	a set of mobile device as a mobile ad-hoc cloud
m_k	mobile device k in the mobile ad-hoc cloud
μ	the average CPU speed of the mobile devices or VMs
τ_n	link delay between client and local mobile device cloud
θ	the average CPU usage
r_i	data transferring time for task i
wm_i	the wireless medium used for offloading task i
l_i	the execution location of task i
α_1, α_2	weight factors used in the general cost model to adjust the user preference
ρ_d	coefficient of channel energy consumption reflecting on execution performance
$B_{channel}$	the bandwidth of the wireless medium, namely WiFi, 3G, and Bluetooth, in MB/s
$C(t_i)$	general overall cost of task i
$D(t_i)$	execution time of task i running on cloud resource
$E(t_i)$	energy consumption of offloading task i
$\Delta E_{channel}$	the energy consumption of task i under certain bandwidth
$\beta_{channel}$	the estimated channel energy consumption per time unit
β_{tail}	wireless medium tail time energy per time unit
T_{tail}	wireless channel active tail time

Problem formulation

Having presented the models of the system, we formulate the decision making problem as to find a solution of selecting where to execute the task and how to offload so that the overall execution time and energy consumption is the lowest among all the cloud resources in the mobile cloud infrastructure based on the current context of the client device. Specifically, given a set of n tasks T, a set of cloud VMs C, a set of cloudlets CL, and a mobile ad-hoc cloud with h mobile devices M, then the overall cost of executing a set of n tasks is

$$C_{total} = \sum_{i=1}^{n} \Delta C(t_i, l_i, wm_i) \tag{3.4}$$

where ΔC denotes execution cost of running task t_i, including execution time and energy consumption. l_i represents the execution location for task t_i, which includes local, mobile ad-hoc cloud M, cloudlet CL, or cloud C. wm_i is the wireless medium used to offload t_i, including Bluetooth, WiFi, and cellular network. Thus, the problem is to provide an offloading decision $\langle l_i, wm_i \rangle$ for $\forall t_i \in T$ to minimize the overall cost.

3.3.2 Cost Estimation Models

fareThe cost model consists of two parts, namely the task execution time denoted by D, and the wireless channel energy consumption denoted by E. Then the general model for overall cost of executing tasks t_i is as follows:

$$C(t_i) = \alpha_1 * D(t_i) + \alpha_2 * \rho_d * E(t_i) \tag{3.5}$$

$$\alpha_1 + \alpha_2 = 1, \alpha_1, \alpha_2 \geqslant 0 \tag{3.6}$$

where α_1 and α_2 are weight factors to adjust the portion of time and energy consumption in the overall cost. α_1 and α_2 are considered to capture user preferences on the factors. If the user is not expert, methodologies like AHP can be used to generate the factors from linguistic values provided by users. The value of execution time and energy consumption in the cost model are normalized in a 0-100 scale with basis normalization upper and lower bound data profiled from real application results. We adopted ρ_d in our model to represent the effect of hardware settings (DVFS levels and wireless channel rate) on the device performance [35]. ρ_d will be set on a 0-1 scale based on the processor DVFS level and wireless channel rate.

Given the general cost models in Equation 3.5, we describe the specific cost models for each type of the mobile cloud infrastructure resource.

Mobile device cloud cost model

First, we model the execution time D for tasks running in the mobile ad-hoc cloud. Given a set of independent tasks $T = \{ t_i \mid 1 \leq i \leq n \}$ and a set of heterogeneous mobile devices $M = \{ m_k \mid 1 \leq k \leq h \}$, execution time D can be obtained by calculating the

earliest finishing time (EFT) among all tasks mapped to the mobile ad-hoc cloud. This independent task scheduling problem is proved to be NP-complete [66]. Therefore the use of heuristics is a suitable approach. For mapping independent tasks to heterogeneous machines, Min-Min heuristic takes less processing time with the result as good as other heuristics [24]. Thus we adopt Min-Min in our cost model.

The Min-min heuristic maps unassigned tasks to available machines. It firstly calculates minimum completion times (MCT):

$$MCT(t_i, m_k) = [min_{1 \leq k \leq h}(CT(t_i, m_k)), \forall t_i \in T] \tag{3.7}$$

where $CT(t_i, m_k)$ is the completion time for task t_i on device m_k. Then task t_i with MCT is selected and assigned to machine m_k, and t_i is removed from T, and the iterations repeat until all tasks are mapped (i.e., T is empty).

In the next step, we model the energy consumption E of the client device. Based on the results from Min-Min, we calculate the device communication energy cost for transferring data to the MANET and receiving the results. Let $B_{channel}$ denote the channel data rate. $t_i = \langle s_i, w_i \rangle$ represents the task ,where s_i is the data need to be transferred and w_i is the workload. The channel energy consumption for transferring data is given as:

$$\Delta E_{channel}(s_i, B_{channel}) = \beta_{channel} * (\frac{s_i}{B_{channel}} + \frac{s_{result}}{B_{channel}}) + \beta_{tail} * T_{tail} \tag{3.8}$$

where $\beta_{channel}$ is the power consumption rate related to the transferring time and β_{tail} is the wireless channel tail time power consumption rate. These two values are adopted from the models in [15].

Let M denote the set of mobile devices in the mobile ad-hoc cloud. μ_k is the CPU speed of device m_k that selected by the MinMin algorithm and θ_k is the average CPU usage. Then the overall execution cost is as follows:

$$C_{t_i} = \alpha_1(\frac{w_i}{\mu_k * \theta_k} + \frac{s_i + s_{i\_result}}{B_{channel}}) + \alpha_2 * \rho_d * \Delta E_{channel}(s_j, B_{channel}) \tag{3.9}$$

Cloud and cloudlet cost models

Let $t_i = \langle w_i, s_i \rangle$ denote a mobile task. There are m VMs available on cloud or cloudlet. μ_{VM} denotes the computing capacity, θ_{VM} denotes the average usage of the VM, and l_{VM} denotes the network latency of the VM. The tasks can be partitioned into subtasks, and VMs on cloud or cloudlet are homogeneous. Hence each task can be evenly partitioned and processed among the machines based on the number of available VMs. Then the cost of running task t_i can be modelled as follows.

$$C_{t_i} = \alpha_1 \cdot \left(\frac{w_i}{m \cdot \mu_{VM} \cdot \theta_{VM}} + \frac{s_i + s_{i\_result}}{B_{channel}} + l_{VM}\right) + \alpha_2 \cdot \rho_d \cdot \Delta E_{channel}(s_i, B_{channel}) \quad (3.10)$$

This cost model is utilized to estimate the task execution cost on cloud and cloudlet VMs by alternating the parameters $\mu_{VM}, \theta_{VM}, l_{VM}$.

Local execution cost

For local cost estimation, we use a history data strategy to reduce the overhead. The local execution time of a method and energy consumption incurred by the device is stored in the database and applied later to the general cost model in Equation 3.5 for comparison. The estimation costs are then used as the input of the decision making algorithm.

3.3.3 Context-aware Decision Making Algorithm

Having shown how to estimate the task execution time and energy consumption with the cost models, we present the algorithm in this section. In order to obtain the lowest execution cost for the offloadable tasks under the context, based on Equations 3.9 and 3.10, the context-aware decision algorithm considers a set of context parameters, multiple wireless medium and mobile cloud infrastructure resources to decide when it is beneficial to offload, which wireless medium is used for offloading and which resources to use as the offloading location.

Wireless medium selection

Most existing offloading frameworks in the literature only consider network speed and energy consumption when they make offloading decisions. Unlike those, mCloud focuses on utilizing multiple types of mobile cloud resources (i.e., cloud, cloudlet and mobile ad-hoc cloud) and wireless mediums based on device context to improve the offloading service availability and performance. Multiple criteria regarding device context including resource availability, wireless medium availability, network congestion, cost, energy consumption, etc. have been considered for making offloading decisions in mCloud. Therefore, we need a multi-criteria decision making approach (MCDM) in the proposed framework.

Among MCDMs, we apply Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) [98] for wireless medium selection considering abovementioned criteria. TOPSIS offers lightweight processing and shorter response time comparing to other MCDMs [223]. It helps reduce the overhead of the proposed offloading decision making algorithm, considering it is running on mobile devices. Moreover, TOPSIS can be easily modified to consider more criteria if necessary, and the complexity remains the same regardless of the number of criteria.

In the mCloud framework, the decision making algorithm considers six criteria related to performance when selecting the wireless interface:

- energy cost of the channel,
- the link speed of the channel,
- the availability of the interface,
- monetary cost (i.e. cost when using mobile data),
- the congestion level of the channel (RTT),
- and the link quality of the channel (signal strength).

Note that for the monetary cost, the algorithm only considers the cost generated by using the mobile data. Other cost such as cloud VM reservation is negligible from the mobile device's perspective as a mobile task generally occupies a negligible time comparing to the cloud VM lifetime.

Table 3.2: Importance scale and definition

Definition	Intensity of importance
Equally important	1
Moderately more important	3
Strongly more important	5
Very strongly more important	7
Extremely more important	9
Intermediate	2,4,6,8

For the alternatives, the algorithm considers Bluetooth, WiFi, and 3G in this system, but more interfaces can be added if new techniques emerge. Then the process of wireless interface selection is as follows.

First, the relative weights for criteria being considered in TOPSIS are obtained by using analytic hierarchy process (AHP) [194]. The pairwise comparison results are presented in a matrix.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{16} \\ a_{21} & a_{22} & \dots & a_{26} \\ \vdots & \vdots & \ddots & \vdots \\ a_{61} & a_{62} & \dots & a_{66} \end{bmatrix}, a_{nn} = 1. a_{mn} = \frac{1}{a_{nm}} \tag{3.11}$$

The pairwise comparisons of six criteria are generated based on the standardized comparison scale of nine levels shown in Table 3.2. Then TOPSIS uses matrix A to calculate the weights of the criteria by obtaining the eigenvector ω related to the largest eigenvalue λ_{max}.

$$A\omega = \lambda_{max}\omega \tag{3.12}$$

Since the output of AHP is strictly related to the consistency of the pairwise comparison, it is necessary to calculate the consistency index [236]:

$$\begin{aligned} CI &= \frac{\lambda_{max} - n}{(n-1)} \\ CR &= \frac{CI}{(n-1) * RamdomIndex} \end{aligned} \tag{3.13}$$

CR, which is the consistency ratio of CI, should be less than 0.1 to have a valid relative

weight output.

After the weights are generated, a evolution matrix consisting of three alternatives and six criteria is created, denoted by $M = (x_{mn})_{3\times 6}$. The values of the criteria are collected at runtime by the *Context Monitor*, and normalized using Equation 3.14.

$$N_{M_{mn}} = \frac{M_{mn}}{\sum_{n=1}^{6} M_{mn}^2} \tag{3.14}$$

Then the weights obtained from AHP method are applied to the normalized matrix $N = (t_{mn})_{3\times 6}$.

$$M_w = \omega_n * N \tag{3.15}$$

where ω_n represents the weight. The best solution and the worst solution are then calculated from the weighted matrix M_w, denoted by $S^+ = \{\langle \min(t_{mn}|m = 1\dots 6)|n \in J^-\rangle, \langle \max(t_{mn}|m = 1\dots 6)|n \in J^+\rangle\}$ and $S^- = \{\langle \max(t_{mn}|m = 1\dots 6)|n \in J^-\rangle, \langle \min(t_{mn}|m = 1\dots 6)|n \in J^+\rangle\}$ respectively, where J^+ is the positive criteria to the cost and J^- is the negative criteria to the cost.

At last, the wireless medium is selected by calculating the Euclidean distance between each alternative and the best and worst solution ${D^+}_m$ and ${D^-}_m$ respectively. and ranking the alternatives by applying a closeness score to the best solution,

$$\begin{aligned} {D^+}_m &= \sqrt{\sum_{n=1}^{6} (t_{mn} - t_{mn}^+)^2} \\ {D^-}_m &= \sqrt{\sum_{n=1}^{6} (t_{mn} - t_{mn}^-)^2} \end{aligned} \tag{3.16}$$

Then rank the alternatives by applying a closeness score R_m to the best solution. The alternative with the highest R_m is selected as the output of the wireless selection algorithm.

$$R_m = \frac{{D^-}_m}{{D^+}_m + {D^-}_m} \tag{3.17}$$

However, one drawback of TOPSIS is its sensitiveness to rank reversal, thus in our system, if a new alternative appears, the algorithm will be triggered to generate the new weights and related matrix.

Algorithm 1 Context-aware decision algorithm

```
1: procedure GETDECISION(context,tasks)
2:     para[] ← context
3:     task[] ← tasks
4:     programProfile ← get method profile
5:     start discovery service and gather resources profiles
6:     local_cost ← estimate execution cost on client device
7:     manet_cost ← estimate execution cost on mobile device
8: cloud using MinMin heuristic
9:     cloudlet_cost ← estimate execution cost on cloudlet
10:    cloud_cost ← estimate execution coston public cloud
11:    check network interface state
12:    if only cell network is available then
13:        check cloud availability
14:        if cloud is available then
15:            decision ← minCost(local, cloud)
16:            return decision
17:        else
18:            return decision(local_execution, null)
19:    else if only WIFI is available then
20:        check cloud, cloudlet and manet availability
21:        decision ← minCost(local, cloud, manet, cloudlet)
22:        return decision
23:    else if only Bluetooth is available then
24:        check manet availability
25:        if manet is available then
26:            decision ← minCost(local, manet)
27:            return decision
28:        else
29:            return decision(local_execution, null)
30:    else
31:        interface ← TOPSIS(context)
32:        if interface is Wifi then
33:            decision ← minCost(local, cloud, manet)
34:        if interface is 3G then
35:            decision ← minCost(local, cloud)
36:        if interface is Bluetooth then
37:            if manet is available then
38:                decision ← minCost(local, manet)
39:            else
40:                return decision(local_execution, null)
```

Decision making

The context-aware decision making algorithm is composed of two main phases, which are summarized in Algorithm 1. The first phase is *estimation* phase (step 2-11), during which the context parameters such as context profiles and wireless interface states are collected from the corresponding modules. Then the cost estimation models calculates the execution cost for each offloading request. In the *selection* phase (step 12-40), the algorithm gives offloading decision based on the available wireless interfaces and the cost estimations. In case there are multiple wireless interfaces available, the algorithm applies TOPSIS model to select the best interface under current context such as data rate, workload size to obtain the best data transfer performance as well as minimum energy consumption. Based on the wireless interface selection result, the algorithm selects the offloading location that has the lowest execution cost. Finally, the algorithm returns the decision pair of $\langle offload\ location, wireless\ medium \rangle$.

The complexity of the proposed algorithm is $O(MNlogN)$, where M is the number of devices in mobile ad-hoc cloud and N is the number of tasks. The first phase of the proposed algorithm uses an improved MinMin (in terms of time complexity) [111] (step 7). Each machine maintains a sorted queue of completion time of all tasks on the machine. It takes $O(MNlogN)$ to construct the queues. In addition, the scheduling takes $O(MN)$ to compare the head of each queue at each scheduling iteration. Thus, the overall complexity is $O(MNlogN)$. Next, for cloud and cloudlet cost estimation (step 9-10), the time complexity is $O(N)$. Therefore, the time complexity of the first phase in the proposed algorithm is $O(MNlogN + N) = O(MNlogN)$. The second phase of the algorithm generates the offloading decision with the time complexity of $O(1)$. Therefore, the time complexity for the proposed algorithm is $O(MNlogN)$.

3.3.4 Failure Recovery

The mobility of mobile devices can incur node failures in the proposed mobile cloud environment, especially the mobile ad-hoc cloud. Although mCloud only considers a one-hop network topology, it is necessary to ensure the consistency of results once the failure occurs. Therefore, we apply a pair of failure detection and recovery policy. The

Algorithm 2 Failure detection policy

```
1:  procedure DETECTION
2:      count[] ← initialized
3:      state[] ← CONNECTED
4:      info[] ← initialized
5:      for checkpoint i to checkpoint n until target time do
6:          Send out confirming message to each node
7:          counter ← counter + 1
8:          for all node i in count do
9:              if Confirmation message received then
10:                 count[i] ← count[i] + 1
11:                 fetch the task running states on node i
12:                 info[i] ← running states
13:         for node i ∈ count[] do
14:             gap ← counter − count[i]
15:             if gap > λ_d then
16:                 state[i] ← FAILURE
17:             gap ← 0
18:         update available node list
19:         Recover(state[], info[])
20:     for node i ∈ state[] == CONNECTED do
21:         if no result returned then
22:             Recover(node i, info[])
```

failures considered by the policy include remote nodes crash, remote nodes out of communication range, and message omissions.

Failure detection

The system adopts checkpointing to detect remote node crash and communication lost in Algorithm 2.

The discovery service inside communication manager periodically broadcasts messages to the remote nodes within the system at each checkpoint and waits for the confirmation message returned by the remote nodes. The discovery service updates a vector that stores the number of confirmation message received from each available node. Then at the end of each checkpoint period, the discovery service calculates the gap between the number of messages received for each node and its own counter. Nodes with same number as the counter or within the distance of λ_d are considered as working, and are

tagged as *CONNECTED*. Nodes that the gap is larger than λ_d are then considered failure, and are tagged as *FAILURE*. Since we only consider one-hop network between the hosting device, MANET devices and cloud VMs, the delivery of the message is not effected by the route change. The failure is mostly likely node crash or dropping out of communication range. In this case, we set λ_d to 3. In case mCloud adopts more complicated network topology in the future, we can adjust the number to fulfil the failure detection requirement.

Moreover, the detecting policy puts the target time on completing the offloading tasks. The target time is related to the estimated execution time provided by the cost models inside decision engine. When the time of checkpoint looping is beyond the target time, the discovery service considers a failure happened in the node that has not returned the result, and proceeds the failure recovery on the tasks from that node.

Recovery

When failures happen, the discovery service fetches the current running states of the tasks on each node at each checkpoint during failure detection. The information includes task ID, the point of failure, and the partial result obtained. If the failure is confirmed by the detection policy, the Discovery Service sends a recovery request to the decision engine. Then the Decision Engine packs all the information of the failed task and choose another available cloud resource node for further execution. If there are no suitable nodes under the current context, the task is then executed locally on the device itself.

3.4 System Design and Implementation

The implementation of our prototype framework is built based on ThinkAir [122]. The system is implemented on Android operating system. We apply VM migration technology using Java Reflection [157] as our offloading method in mCloud to minimize the modifications of the existing or developing mobile applications. The system and programming APIs are implemented as a library for the Android application developers. Android x86 system[92] is deployed on the cloud and cloudlet VMs. In this section, we explain in details the design and implementation of mCloud and programming APIs.

```
public class Example {
 private int expA;
 private int expB;
 @OFFLOAD
 public int solve(){
  return ;
 }
 public void otherMethods(){
 }
}
```

Listing 3.1: Offloading example

3.4.1 Android x86

In order to leverage the cloud resources, the framework needs to offload the mobile tasks from the ARM-based system application to an x86-based cloud VM. Android x86 is an open-source project to port Android operating system to an x86 or AMD host. It provides major functions of an Android device on desktop system like Windows. As a result, our proposed framework can fit into most of the existing and developing Android applications and cloud services without modifications on the program. We deployed the Android x86 system on Virtualbox virtual machines running on commercial public cloud services.

3.4.2 Offloading Method

We use Java reflection as mCloud's offloading method. Java reflection allows the program to inspect the available Java classes, methods, interfaces and their properties within the system or itself at runtime [14]. We can manipulate programs with certain classes, retrieve related properties like parameter types and fields, and invoke the methods of other programs remotely. Our proposed mCloud framework implements the offloading and remote execution by using Java reflection and Java annotations.

We provide a simple annotation @OFFLOAD for developers to annotate the methods to be considered for offloading. The annotated methods will be processed at the compile time via Java Reflection. Listing 3.1 shows an example of using Java annotation for offloading. At runtime, the application program can inspect and invoke method *solve* in the offloading example.

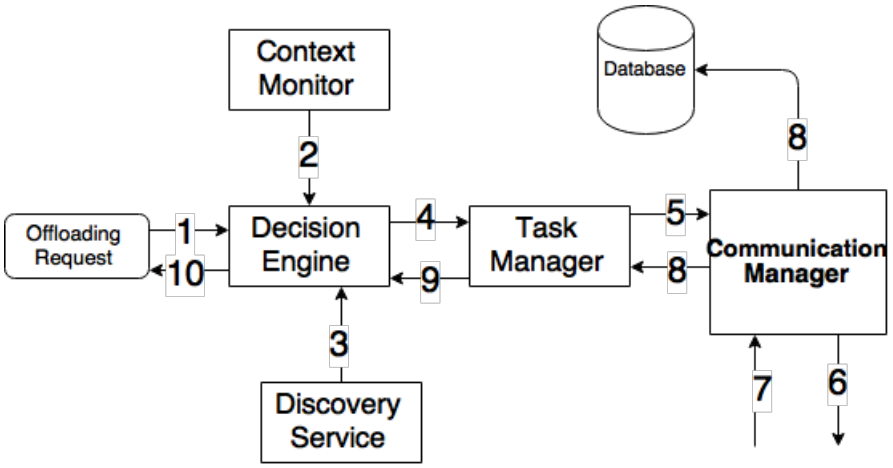


Figure 3.2: Execution dataflow: 1) sending offloading request to the decision engine, 2) collecting context parameters from context monitor, 3) get information of available cloud resources, 4) Task Manager starts once the decision is made to offload, 5) Communication Manager divides the jobs into subtasks for parallel processing, 6) Offload to cloud resources for remote execution, 7) pause until receiving the result, 8) aggregate results from parallel processing and store the result of execution time and energy consumption in database, 9) and 10) send result back to device for presentation

3.4.3 Execution Environment

The execution dataflow is depicted in Figure 3.2. Five main components are implemented, namely Context Monitor, Decision Engine, Task Manager, Communication Manager, and Discovery Service. These components constitute the execution environment for the code offloading tasks.

Discovery Service

The Discovery Service starts when the application is opened on the device. It detects two different types of resources: the available Android x86 VMs in the remote, and mobile devices in the proximity. The Service first contacts a root server on the cloud VM and retrieves a list of available Android x86 VMs and their IP addresses on the public cloud and local cloudlets. Meanwhile, the Discovery Service starts an new thread in the background, which establishes a WiFi hotspot to let the mobile devices nearby connect to the client device. Then the Service can simply ping within a certain IP address range to detect the available mobile devices. The benefit of this approach is that in most cases when the public cloud services are not available, for example at a disaster recovery site, using

```
public void networkProfiler(){
 networkStateReceiver = new BroadcastReceiver(){
  public void onReceive(){
  netInfo=connectivityManager.getActiveNetworkInfo();
  networkType = netInfo.getTypeName();
  }}}
```

Listing 3.2: Profiler implemented with BroadcastReceiver

hotspots to form an ad-hoc network is stable and easy to establish.

The service will search devices in *periodical searching* strategy to keep the available mobile device list updated. Additionally, the Service also considers Bluetooth connection in Discovery Service. It will activate the Bluetooth module on the client device when the service starts at the beginning and initialize a Bluetooth discovery session that detects other Bluetooth capable devices in the range. The hosting mobile device then gets a list of bonded devices and other available devices for connection. All the information of the mobile devices discovered by the system will be stored for the further decision engine evaluation.

Context Monitor

Device profiler, network profiler and program profiler are implemented in the Context Monitor. The Context Monitor is designed to collect context data in an on-demand monitoring strategy in order to reduce the overhead. Hence, the profilers are implemented with BroadcastReceiver[3] to receive the context data only when the context changes. Listing 3.2 shows an example of the network profiler. The profiler in the example will detect the current active network connection type when it is changed.

Communication Manager

The communication manager extracts the information of the IP address of the offloading location, offloading task ID, method name, input parameters, and wireless medium type being used generated by task manager. Then it starts a new asyncTask[4] in the background to initialize the connection to the selected cloud resource. The communication manager

[3] An Android class that can receive broadcasts across the applications

[4] An Android class that allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

```
private void remoteExecution(InputStream objIn){
 obj = objIn.readObject();
 name = objIn.readObject();
 parameters = objIn.readObject();
 paraTypes = objIn.readObject();
 //Get the class
 class = obj.getClass()
 //construct class in remote host
 constructor = class.getConstructor();
 instance = constructor.newInstance();
 //Get the offloading method
 runMethod = class.getDeclaredMethod(name,
  paraTypes);
 runMethod.setAccessible(true);
 //invoke the remote method with input parameters
 result = runMethod.invoke(class,parameters);
}
```

Listing 3.3: An example of remote execution

detects if connected cloud resource has the application files and relevant libraries, and sends missing files. Then the Manager waits until the asyncTask collects the results.

Decision Engine

Upon receiving the offloading request, the Decision Engine first fetches all the information from Context Monitor and Discovery Service. Then it passes all the context parameters and available device information to the decision algorithm for evaluation. The decision is packed into an array as execution location, machine IP address, offloading method name, input parameters, and wireless channel for communication.

Once the Decision Engine makes the offloading decision, the framework starts the remote execution. Listing 3.3 gives an example of the remote execution. The remote cloud resource first unpacks and get the parameters regarding the execution environment, such as method name, input parameters, and return types. Then it use Java reflection to create a new instance of the offloaded class. Last, it invokes the annotated method to execute and return the results to the hosting mobile device. During the remote execution, the hosting mobile device is on hold until all the results are returned from the cloud resources.

Table 3.3: Criteria weights for TOPSIS

Criteria	Weight	CR
Power Consumption	0.180	
Bandwidth	0.130	
Cloud Resource Availability	0.514	0.052
Congestion Level	0.081	
Signal Strength	0.062	
Cost	0.033	

3.5 Performance Evaluation

3.5.1 Experiments Settings

To the best we know, there are no available standard testbeds that are suitable for the performance evaluation. Thus, to evaluate the effectiveness of mCloud and offloading scheme on different kinds of mobile applications, we implement two Android applications. The applications represent two different types of tasks, one is small file size with high computation, and the other one is big file size with high computation. For the first type, we implement an application that performs math operations based on the input data, and for the second type we implement a face detection application.

We deploy the applications on one HTC G17, one Samsung I997, and one Nexus 5, which form a device cloud for the experiments. One Android x86 clone is installed within VirtualBox on an Intel i5 laptop serving as cloudlet, the emulated CPU speed was adjusted from VirtualBox to match the processing speed of cloudlet. Two Android x86 clones are set up in an Amazon EC2 t2.medium instance. Moreover, we use PowerTutor [250] to monitor the energy consumption of CPU and communication. Unrelated applications, background services (e.g. GPS, audio, etc.) and screen of the mobile devices are shut down during experiments. The relative weights for criteria in TOPSIS model is generated based on the system priority. Due to the concern on the processing delay and energy consumption, we set the order of priority set for the six criteria using the pair comparison method in AHP: resource availability > power consumption > bandwidth > channel congestion level > signal strength > monetary cost. Based on this assumption we calculate the weights from AHP and the results are shown in Table 3.3. The consistency

Table 3.4: Experiment scenarios

No.	Application	Scenario	User Preference
1	Math application, face detection	stable device context, test performance against baselines	time_sensitive energy_sensitive time_energy
2	Math application	unstable mobile cloud resource availability, stable network (Table 3.7)	time_energy
3	Math application	stable mobile cloud resource availability, unstable network (Table 3.8)	time_energy

ratio [5] value is 0.052 (less than 0.1), thus the weights are valid.

We conducted three sets of experiments. A summary of the scenarios is listed in Table 3.4. In the first scenario, two applications are executed in a stable device context (i.e. all mobile cloud resources and wireless mediums are available and stable) to evaluate the performance of mCloud in terms of time and energy consumption, under three user preference policies. Then we compare them with the baselines of *local_only*. In the second scenario, we conduct experiments under multiple cases of mobile cloud resource availability, while in the third test scenario, mCloud is tested under unstable network conditions. These results are then compared with the existing work ThinkAir. In summary, the second and third set of experiments aim to demonstrate the advantages of mCloud in an unstable mobile cloud context.

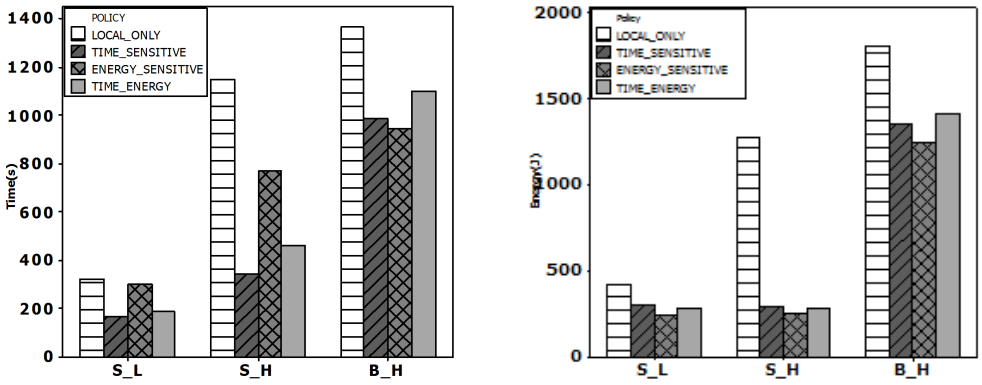
3.5.2 Results and Analysis

We run the two implemented mobile applications with 500 input tasks respectively under three offloading policies, namely *time_sensitive*, *energy_sensitive*, and *time_energy*. Then the results are compared with the baselines that runs workload in offloading policy *local_only*. The characteristics of the generated workloads is listed in Table 3.5. Workload S_L and S_H are generated by the calculation application, and workload B_H is generated by the face detection application. Each measurement result is calculated by the average of 10

[5] Calculated by Equation 3.13

Table 3.5: Workload for experiments

Workload	Average data size (byte)	Average Android bytecode instructions (MI)	Number of tasks
S_L	725	5.8	500
S_H	650	24	500
B_H	3000	29.5	500



(a) Time(s) (b) Energy(J)

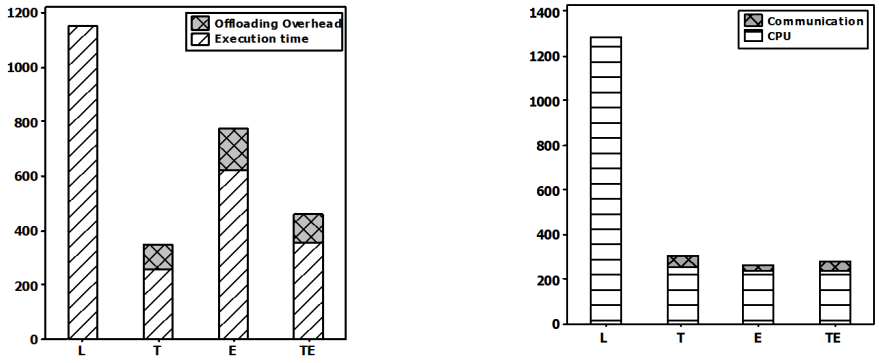
Figure 3.3: Overall time and energy consumption for each workload under different policies

trails. Figure 3.3a and Figure 3.3b compare the execution time and energy consumption respectively of each workload under three offloading policies, which are time sensitive, energy sensitive and time energy combination. Table 3.6 lists the proportion of the tasks allocated to the multiple cloud resources in mCloud's mobile cloud environment.

As shown in Figure 3.3a, for workload S_L, the execution time is reduced by around 55% under *time_sensitive* policy comparing to *local_only* policy. 75.4% of the tasks are scheduled by the decision engine to the cloudlet server (shown in Table 3.6) that has a much lower network latency than public cloud. In Figure 3.3b, the energy consumption for workload S_L is reduced by 55.6% under *energy_sensitive* policy, which has the best performance among all policies. 84.2% of the tasks are scheduled to the MANET due to the low energy consumption on data transferring via Bluetooth. For the *time_energy* policy, the result shows in Table 3.6 that 9.6% tasks are executed in local, 70.4% tasks in cloudlet, and 20% in public cloud, with the consideration of network condition and available cloud resources.

Table 3.6: Proportion of tasks mapped at each location under different policies (T: *time_sensitive*, E: *energy_sensitive*, TE: *time_energy*)

Workload	Policy	Local	Manet	Cloudlet	cloud
S_L	T	1	0	75.4	23.6
	E	15.8	84.2	0	0
	TE	9.6	0	70.4	20
S_H	T	0	0	31.6	68.4
	E	0	82.1	17.9	0
	TE	0	0	79.3	20.7
B_H	T	41.2	0	58.8	0
	E	60.8	39.2	0	0
	TE	33.5	12.6	53.9	0

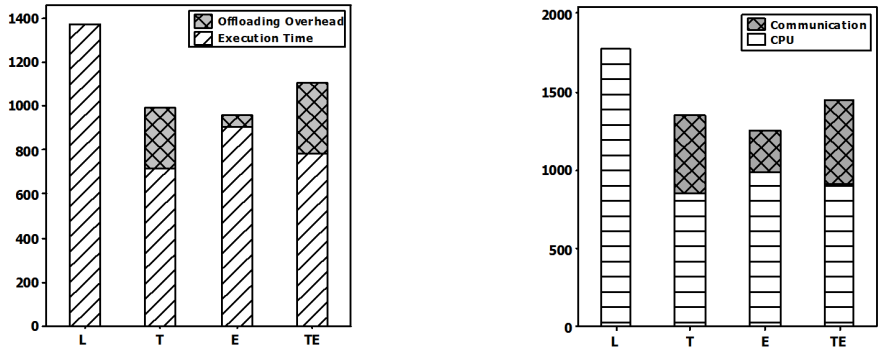


(a) Time(s) (b) Energy(J)

Figure 3.4: Task processing time and offloading overhead of workload S_H under local_only(L), time_sensitive(T), energy_sensitive(E) and time_energy(TE) policies

We can also observe similar results that, for workload S_H, mCloud gives the best performance by achieving 65% of time reduction under *time_sensitive* policy and 70% of energy reduction under *energy_sensitive* policy. For workload B_H, mCloud achieves 25% of time reduction and 30% of energy reduction on average. The experiment results show the mCloud is most beneficial to the tasks that have low data size and high computation.

Figure 3.4 and Figure 3.5 break down time and energy consumption of workload S_H and B_H respectively. For workload S_H in Figure 3.4a, the offloading overhead under different policies is around 20% on average, while the offloading overhead of workload B_H in Figure 3.5a is much larger. The difference of offloading overhead between these two workloads is due to the time consumed in transferring the data. The energy consumption of communication is fairly small since the workload S_H includes tasks with



(a) Time(s) (b) Energy(J)

Figure 3.5: Task processing time and offloading overhead of workload B_H under different policies

small data sizes (Figure 3.4b). On the contrary, workload B_H contains tasks with large data sizes that increase the energy consumption of transferring the data (Figure 3.5b). The communication of workload S_H costs around 13.6% of the total energy consumption on average, while for workload B_H it costs more than 30% on average. Figure 3.5a shows the overall time under *time_sensitive* is greater than *energy_sensitive*, while the execution time under *time_sensitive* is smaller. This is because the tasks under *time_sensitive* policy were scheduled among local and cloudlets via a WiFi public access point on our site with long latency, and tasks under *energy_sensitive* policy were scheduled among local and Manet via Bluetooth and WiFi-direct, which has almost no network latency. Consequently, for workload B_H that has large data size to transfer, although the task execution time is lower, more time spent on waiting for results and transmission under *time_sensitive* policy. The overall time will be shorter under *time_sensitive* when using a lower latency access point.

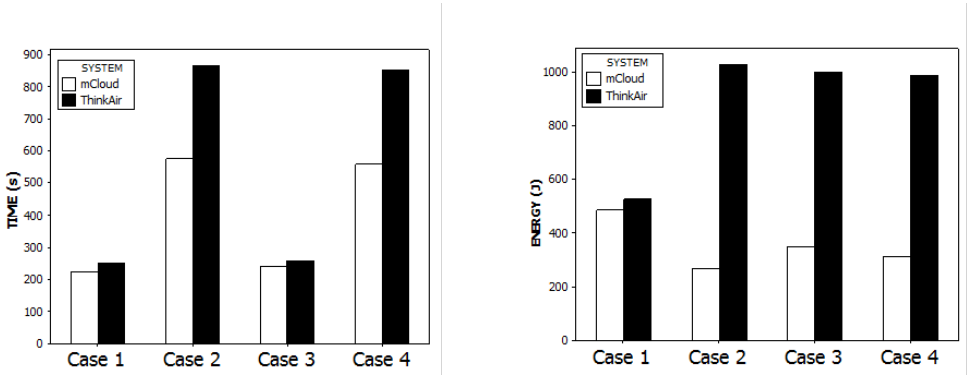
Furthermore, to explore the performance of mCloud in terms of execution time and energy consumption under unstable contexts, we conduct the experiment using workload S_H with different combinations of available cloud resources and network conditions under time and energy combined policy, and compare performance with ThinkAir.

First, we evaluate the performances in different available cloud resources conditions while the network condition is stable. The test cases are listed in Table 3.7.

Case 1 indicates all resources are available while case 2 to case 4 represent the absence of public cloud service, cloudlet service, and nearby wireless mobile ad-hoc network

Table 3.7: Number of each type of cloud resources

Case	Cloud VM	Cloudlet	MANET
1	2	2	3
2	0	2	3
3	2	2	0
4	0	0	3



(a) Time(s) (b) Energy(J)

Figure 3.6: Performance under available resource changing conditions

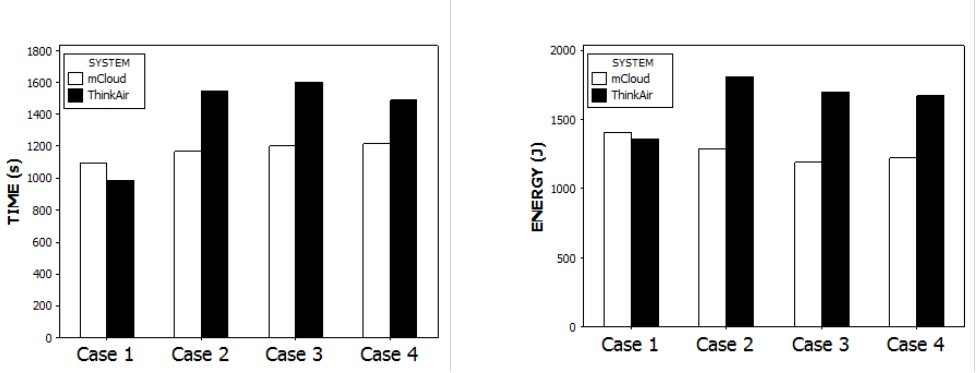
respectively. The results are illustrated in Figure 3.6.

Figure 3.6a and Figure 3.6b show the execution time and overall energy consumption of the workload in each case for our proposed system and ThinkAir respectively. In Figure 3.6a, it shows that mCloud outperformed ThinkAir in terms of time saving in all the four cases. Especially in case 2 and case 4, due to the unavailability of public resources, ThinkAir chooses to run all the tasks locally on the device, while mCloud offloaded part of the tasks to the mobile clones on cloudlet and the nearby mobile device cloud, which conserved around one third of the time ThinkAir took for execution. The similar result can be observed for energy consumption of the two systems in Figure 3.6b. The results show that our proposed system can provide code offloading services when the mobile cloud environment is short of public cloud resources and help conserving execution time and battery, unlike many existing mobile cloud frameworks.

Second, we compare mCloud with ThinkAir under unstable network condition. The experiments are conducted by executing workload B_H under changing network context. We alter the bandwidth of Internet connection and 3G connection to simulate the changing network condition in the real world. Table 3.8 lists the four test cases. The max-

Table 3.8: Average network speed for each test case

Case	Wifi (MB/s)	3G (MB/s)	Bluetooth (MB/s)
1	14.3	3.5	2.8
2	0	3.2	3.0
3	0	0	2.2
4	1.7	3.3	2.5



(a) Time(s) (b) Energy(J)

Figure 3.7: Performance under different network conditions

imum bandwidth of our testing devices' WiFi connection was 14.3 MBps, the average 3G connection speed was 3 MB/s. The Bluetooth speed was the same throughout the experiment. When the speed was set to 0, it represents the unavailability of the corresponding wireless channel. In order to alter the bandwidth of the network, we set up a virtual access point on the laptop and connect DummyNet [28] to the virtual AP to manage the network bandwidth, latency, etc.

Case 1 represents the scenario where all the wireless channels are available and operating at full speed. Case 2 represents the scenario where WiFi connection is not available. Case 3 represents neither WiFi nor 3G is available. Case 4 represents the WiFi connection speed drops from full speed to low speed that is slower than 3G and Bluetooth. Figure 3.7a and Figure 3.7b show the execution time and overall energy consumption of the workload in each case for our proposed system and ThinkAir respectively.

As illustrated in Figure 3.7a and Figure 3.7b, test case 1 shows that mCloud has close performance as ThinkAir in terms of execution time and energy consumption. For case 2, 3 and 4, mCloud has around 20% performance gain comparing to ThinkAir. For test case 2, when only mobile data and Bluetooth are available, mCloud schedules the offloading

tasks to either MANET through Bluetooth or local due to the consideration of monetary cost. The result of test case 4 shows that when the network is unstable or slow, mCloud can save more execution time and energy than ThinkAir because of the multiple cloud resources and context being considered in mCloud.

3.6 Summary

In this chapter, we proposed a context-aware offloading decision algorithm that takes into consideration context changes (e.g. network conditions and heterogeneous mobile cloud resource) to provide decisions on wireless medium and mobile cloud infrastructure resources to utilize at runtime. It also provides a general cost estimation model for mobile cloud infrastructure resources to estimate the task execution cost including execution time, energy consumption. The models can be easily modified for the new cloud resources. A prototype system designed and implemented that considers three types of cloud resources (mobile device cloud, cloudlet and public clouds) and a decision engine that runs the proposed algorithm and related cost estimation models. The evaluation of mCloud is presented, and results showed that the system can provide offloading decisions based on the current context of mobile devices to lower the cost of execution time and energy.

However, a drawback of the proposed context-aware offloading decision algorithm is that it only makes decisions from a single view of the mobile device. Since the heterogeneous mobile cloud environment we proposed leverages a shared resource pool where all the devices within the environment can offload tasks and execute the offloaded tasks from others, an optimal task offloading schedule with load balancing consideration is needed to provide an overall better performance. Therefore, in the next chapter, we propose an online task offloading and scheduling algorithm for the task offloading in heterogeneous mobile clouds.

Chapter 4

Execution Optimization for Task Offloading and Scheduling

A mobile device within the heterogeneous mobile cloud environment can both offload mobile tasks and run the offloaded tasks. The offloading decision making and tasks scheduling among heterogeneous shared resources in mobile clouds are becoming challenging problems in terms of providing global optimal task response time and energy efficiency with load balancing. In this chapter, we address these two problems together as an optimization problem. The proposed optimization problem considers unique contexts of mobile clouds such as wireless network connections and mobile device mobility, which makes the problem more complex. We propose an offline optimal solution based on mixed integer programming model for making the offloading decisions and scheduling the offloaded tasks among the shared computing resources in heterogeneous mobile clouds to minimize the global task completion time and provide load-balanced schedules. To solve the problem in real-time, we further propose an online scheduling algorithm named OCOS based on the rent/buy problem and prove the algorithm is 2-competitive. Performance evaluation results show that the OCOS algorithm can generate schedules that have around 2 times shorter makespan than conventional independent task scheduling algorithms.

4.1 Introduction

MOBILE computing technologies developed enormously in recent years, mobile devices (e.g., smartphones, tablets, and wearable devices) have been involved as part of people's daily activities [56]. Mobile applications such as cognitive applications (e.g., optical character recognition, face detection) and augmented reality are gaining popularity among mobile device users. These applications typically require intensive

This chapter is derived from: **Bowen Zhou**, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya, "An Online Algorithm for Task Offloading in Heterogeneous Mobile Clouds," *ACM Transactions on Internet Technology*, Volume 18 Issue 2, Article No. 23, March 2018.

computation as well as considerable energy consumption. However, compared to desktop computers, mobile devices still provide a relatively inferior performance in terms of processing capacity, memory, storage capacity and battery life to support long-term services. As a result, the gap between resource-constrained mobile devices and computing intensive applications has posed a significant challenge.

Previous works [37, 125] have seen limits of only offloading to cloud since wireless networks can cause the performance bottleneck. To overcome this challenge, it is of interest to provide mobile cloud service users with alternative computing paradigms. Due to the improvement in short range wireless networks and mobile devices, mobile cloud computing has evolved beyond the onefold public cloud resource into a heterogeneous network of computing resources.

As illustrated in Figure 1.1, the heterogeneous mobile cloud (HMC) environment contains different types of computing resources such as public clouds, private clouds, cloudlets [200], and mobile ad-hoc networks (MANET). For each mobile device, it is important to make the offloading decision based on its context such as network conditions, load of other machines, and mobile device's own constraints (e.g., mobility and battery). Moreover, to achieve a global optimal task completion time for tasks from all the mobile devices, it is necessary to devise a task scheduling solution that schedules offloaded tasks in real time.

In this chapter, we jointly investigate the mobile code offloading problem and task scheduling problem for offloaded tasks as a mobile code offloading and scheduling problem (MCOSP). MCOSP not only makes decisions of whether, when and how to offload tasks for each mobile device, but also aims to schedule offloaded tasks among shared machines in heterogeneous mobile clouds. MCOSP takes into consideration the constraints of the shared computing resources (i.e., load of the resources, network conditions, and limits of battery lifetime) when scheduling, such that the overall task completion time is minimized. Our work considers a common task scheduling model where the mobile device within the heterogeneous mobile cloud environment generates independent tasks at arbitrary time, and one machine may only process one task at a time. The heterogeneity in both tasks and machines makes MCOSP an NP-hard problem [76]. We study the optimization of MCOSP in both offline and online cases. Table 4.1 lists the acronyms used in

Table 4.1: List of acronyms

Acronyms	Description
HMC	Heterogeneous Mobile Clouds
MCOSP	Mobile Code Offloading and Scheduling Problem
OCOS	Online Code Offloading and Scheduling algorithm
LCSD	Low Computation Small Data Size workload
HCSD	High Computation Small Data Size workload
HCLD	High Computation Large Data Size workload
VM	Virtual Machine
OCR	Optical Character Recognition
OLB	Opportunistic Load Balancing algorithm
LO	Mobile Local Only
CO	offloading to Cloud Only

this chapter. The **key contributions** of this chapter are:

- First, we propose the models of computation, energy consumption, monetary cost and time-to-failure for mobile devices that represent the heterogeneity of the proposed heterogeneous mobile cloud environment.
- Second, we formulate MCOSP as a mixed integer nonlinear programming problem based on the models of the proposed heterogeneous mobile clouds and provide an analysis on its hardness. Then we transform it into a mixed integer linear programming formulation using linearization and solve the problem using the branch-and-bound algorithm.
- Finally, to provide a practical solution for the problem, an online real-time scheduling algorithm for MCOSP based on a generalization of rent/buy problem framework is proposed to obtain competitive schedules for large workloads. Experimental results demonstrate that the proposed online algorithm OCOS generates 2-competitive makespan on average comparing to offline optimal solution in terms of scheduling performance, and scales well in terms of offloading gains when the number of mobile users in the system increases.

The remainder of the chapter is organized as follows. In Section 4.2, the system models for task scheduling are introduced, followed by a description on MCOSP and the MIP-based formulation in Section 4.3. Then we proposed the online solutions and its

competitive analysis in Section 4.4. The experiment settings are explained and the performance evaluation results are discussed in Section 4.5. Finally, we conclude the chapter in Section 4.6.

4.2 System and Cost Models

4.2.1 Motivation Example

A group of overseas tourists are travelling at a local museum. Since they do not speak the local language, an optical character recognition (OCR) application is installed on smartphones and tablets to help them read museum exhibit descriptions. The OCR application takes photos of the descriptions, processes to extract the text from the photos, and sends extracted texts over the Internet to its back end services in cloud for translation. However, some mobile devices do not have enough computing capacity to perform the text extraction, and some other mobile devices do not have an Internet connection to get the translation. In order to enable every group member to use the OCR application on his or her device, mobile devices, including smartphones, tablets and laptops, and public cloud VMs running back end services, form a network of shared computing resources (i.e., HMC). The individual mobile device that lacks resources is able to utilize the shared computing and network resources by offloading tasks to other devices or the cloud.

The challenge in the scenario is deciding whether, where and how to offload a mobile task, as well as scheduling the offloaded tasks with load balance among the shared resources to achieve optimal task completion time.

4.2.2 Resource Models

Suppose a set of heterogeneous machines $m \in M$ are connected via different wireless technologies. M consists of three types of computing resources: virtual machines on a single public cloud service, nearby cloudlets, and mobile devices in proximity. The heterogeneity of each machine m is modelled in terms of computing and network capability. The model is given as $m \triangleq \langle \mu, l, r, \theta, PR^{active}, PR^{idle}, E^{total}, t^{join}, t^{leave}, C^{budget} \rangle, \forall m \in M$, where μ is the processing speed of machine m, l is the network latency between the client

Table 4.2: Symbols of the HMC modeling

Symbol	Description
M	the set of heterogeneous machines such as cloud VMs, cloudlets and MANET
S	the set of independent tasks to be scheduled
m	machine m in HMC environment
i	mobile task i
f_i	the data size of task i
ϖ_i	the computation workload of task i
t_i^{arrive}	the arrival time of task i
μ	the processing speed of the machines
l	network latency of the HMC network
r	the charge rate per time unit for cloud instance in M
θ	the proportion of battery energy provided by mobile device in M
ν_m	the energy consumption rate of CPU on machine m
PR^{active}	energy consumption per time unit when the machine is in the active mode
PR^{idle}	energy consumption per time unit when the machine is in the idle mode
E_m^{total}	the total battery energy of mobile device m
t_m^{join}	the time when machine m join the HMC network
t_m^{leave}	the time when machine m leave the HMC network
$T_{i,m}^{exec}$	the processing time of task i on machine m
$T_{i,ch}^{trans}$	the data transferring time of task i using wireless channel ch
$E_{i,m}^{exec}$	the energy consumption for executing task i on machine m
$E_{i,m}^{trans}$	the energy consumption for transferring task i to machine m using wireless channel ch
$x_{i,m}$	the binary variable indicating task i is assigned to machine m in HMC
w_i, c_i, b_i	the binary variables indicating the wireless medium used to offload task i
$o_{i,j,m}$	the binary variable indicating whether task i is scheduled before task j on node m
$t_{i,m}^{start}$	the real variable representing the start time of task i on machine m
$t_{i,m}^{finish}$	the real variable representing the finishing time of task i on machine m
$\overline{T}$	a constant greater than the worst case makespan
R_i	renting cost of task i
$B_{i,m}$	buying cost of dispatching task i to machine m
C_m^{budget}	the monetary budget of machine m
C_i^{trans}	the monetary cost of transferring data of task i via mobile data
C_i^{VM}	the monetary cost of executing task i on cloud VMs

device and m, r is the charge rate per time unit for cloud instance, θ denotes the proportion of energy machine m will provide to execute offloaded tasks. M represents the set of machines that form the shared resource pool. Note that θ is an exclusive property for mobile devices. The value of this metric for other types of computing resources is set to 0. PR^{active} represents the energy consumption rate per time unit of the mobile device in active mode, and PR^{idle} represents the energy consumption per time unit in idle mode. E^{total} is the total battery energy of machine m. The mobility of mobile devices are represented by the joining time and leaving time respectively as t^{join} and t^{leave}. C^{budget} denotes the monetary budget of machine m for using the mobile cloud service and cellular

network data usage.

Mobile devices are not always available due to the mobility of users, and the available time of the machines in HMC has a vital impact when scheduling mobile tasks. To enable a tractable analysis, we first assume that network settings (i.e. signal strength, network speed and latency) remain unchanged throughout the time when the mobile device is within HMC environment. This is reasonable considering a group based environment such as offices or group travelling.

In each proposed HMC network, a task scheduler is deployed to receive task offloading requests from mobile devices within the network and schedule the tasks based on the proposed scheduling algorithms. To reduce scheduling overhead, the scheduler is placed on the nearby cloudlet to reduce network latency. In case the cloudlet is not available or in failures, the scheduler is placed on the VM on the public cloud as a backup. The proposed HMC environment is enabled by the mCloud framework we proposed in chapter 3. A summary of the notions used in the system models is listed in Table 4.2.

4.2.3 Workload Model

Based on the experimental results in Chapter 3, only tasks with high computation and small data input size are beneficial from mobile cloud offloading (e.g. OCR and GPS-based applications), and we call it offloading tasks. Consider an application is composed of offloading tasks and other tasks, the proposed algorithm only considers offloading tasks for scheduling. Therefore, each application can only submit the offloading tasks to the scheduler. We assume that the tasks evaluated at the scheduler are independent and non-preemptive of each other. We define a 3-tuple for offloading tasks $i \triangleq \langle f_i, \varpi_i, t_i^{arrive} \rangle$, where f_i is the data size of the task i to be offloaded, ϖ_i denotes CPU cycles needed to complete task i, and t_i^{arrive} represents the arrival time of task i at the scheduler. The mobile device user can adopt the methods proposed by Cuervo et al. [52] and Chun et al. [43] to obtain the value of f and ϖ. Tasks are generated at arbitrary time. Due to the concern of communication overhead, once offloading tasks are generated, only the information of 3-tuple model is sent to the central scheduler at the stage of scheduling.

4.2.4 Computation and Communication Models

The execution time and energy consumption required to execute a task are two major factors considered in the process of making offloading decision. CPU and wireless transmitter are two major energy consumption sources for mobile devices in mobile cloud computing. Typically the power consumption of these hardware components depends on their operating state (i.e., utilization, data rate, etc.). Without loss of generality, we assume that the CPU operates at full utilization when executing mobile tasks and the data rate of the wireless channel is constant and stable. In our model, the scheduler takes into account two metrics: the computation time, and the energy consumption of processors and transmitters.

Suppose that an offloading task i is waiting to be scheduled to M. The computation execution time of task i on machine m is composed of CPU computation time and memory I/O access time. Since memory access is tightly coupled with the type of application and instructions executed and can vary on different hardware architectures, the model takes on a high level abstraction of memory access based on cache misses model proposed by Fan et al. [65].

$$T_{i,m}^{exec} = \frac{\omega_i}{\mu_m} + t_{access} * N_{miss},$$

where t_{access} and N_{miss} represent memory access time and number of cache misses respectively. These two parameters can either be obtained from hardware specifications or from existing performance counters on many modern processors [65]. If machine m is not the one that generates task i, a task offloading occurs with extra cost in terms of time and energy required for data transmitting. The data transmission time is given as

$$T_{i,m}^{trans} = \frac{f_i}{BW_{ch}},$$

where BW_{ch} denotes the network speed of the wireless channel used to offload data. The computation execution time for an offloaded task is the sum of $T_{i,m}^{exec}$ and $T_{i,m}^{trans}$. Note that the time and energy cost for sending computation results back to mobile device user are neglected, due to the fact that the data size of results for the considered applications (i.e. cognitive applications) is much smaller comparing to the offloaded data (i.e., input data,

and application code).

Similar to computation models, the energy consumption model consists of the execution of offloaded tasks, and energy consumed by data transmission if offloading occurs. The energy consumption model for task i executing on machine m is given as

$$E_{i,m}^{exec} = \nu_m \frac{\varpi_i}{\mu_m},$$

where ν_m represents the energy consumption rate of m executing tasks based on the processor's frequency [233]. $E_{i,src}^{exec}$ is the energy consumption for machine m if the scheduler decides to execute task i locally. For the computation offloading, the amount of energy incurred by data transfer, $E_{i,m}^{trans}$, depends on the size of the data and the wireless medium that carries the transmission.

$$E_{i,m}^{trans} = \rho_{ch} \frac{f_i}{BW_{ch}},$$

where ρ_{ch} denotes the energy consumption rate of the wireless medium applied, and BW_{ch} is the network speed. Accordingly, the energy consumption of task i executing locally is $E_{i,src}^{trans}$.

4.2.5 Monetary Cost

We assume that each individual mobile device has a monetary cost budget set by the mobile user for using the proposed mobile cloud services. The monetary cost is generated by transferring data via cellular networks and using public cloud services. The scheduler needs to make offloading decisions without violating budget constraints. In our system model, the data transferring cost of offloading task i is calculated as $C_i^{trans} = \kappa(f_i + f_{res})$, where κ is the cost per megabyte of cellular networks, and f_i, f_{res} denotes the data size of task i and computation results respectively.

The cost of using public cloud services is based on the type of service and the usage. The spending on running task i on cloud VM m is calculated as $C_i^{VM} = r_{vm} \frac{\varpi_i}{\mu_{vm}}$, where r_m is the cost per time unit of cloud instance m.

4.2.6 Failure Model

Mobile devices are unreliable due to its mobility and unstable wireless network connections. The disconnections of mobile devices from the mobile cloud environment can cause failures in execution of offloaded tasks. Therefore, considering the availability of machines is important for making offloading decisions. Note that the study of mobility patterns of mobile devices is not the focus of this chapter. Instead, the system model adopts the time-to-failure metric to represent the availability of the machines. Time-to-failure represents the time between the machine joining (i.e. t_m^{join}) and being disconnected from the shared resource pool (i.e. t_m^{leave}).

To obtain the time-to-failure of mobile devices, Weibull distribution is adopted since it can well represent the case where failure rate is constant as well as the case where failure rate increases or decreases as time goes by [129], and is shown to be useful for modelling lifetime data in engineering sciences [128]. The PDF of Weibull distribution is given by:

$$f(t) = \frac{\beta}{\eta}(\frac{t}{\eta})^{\beta-1}e^{-(\frac{t}{\eta})^{\beta}},$$

where β is the shape parameter and η is the slope parameter of the distribution. Time-to-failure of each machine can be randomly obtained from the inverse function of Weibull distribution. t_m^{leave} of machine m can be calculated by padding the time-to-failure after t_m^{join} of the machine.

4.3 Mobile Offloading Scheduling Problem Formulation

In our system model, mobile tasks, machines, and wireless networks are all considered heterogeneous. To guarantee a consistent user experience for all mobile cloud service users, we target on the balanced utilization of the shared computing resources with the minimized overall task completion time, considering the heterogeneity and constraints of the HMC environment. In such case, task completion time for individual user may not be optimal, but for all the service users, the task completion time of all mobile tasks submitted by mobile devices is minimized to ensure the balanced and fair use of the shared computing resources.

4.3.1 Mixed Integer Programming Based Offline Optimal Formulation

The mobile code offloading and scheduling problem (MCOSP) aims to decide on which machine to execute each mobile task, as well as which wireless medium to use if an offloading is required. Furthermore, it needs to devise an optimal schedule for the mobile task execution with minimum makespan, subject to constraints from the proposed HMC environment.

Based on proposed system models, we presented a mixed integer programming (MIP) based problem formulation for MCOSP. In the MIP formulation, 5 binary variables and 2 continuous variables are defined to devise the optimal solution. Note that the term *machine* is used to represent any types of computing resources in the proposed HMC environment.

Let $t_{i,m}^{start}$ and $t_{i,m}^{finish}$ be the starting time and finishing time of task i executing on machine m respectively. The binary variable $x_{i,m}$ represents whether task i is scheduled on machine m.

$$x_{i,m} = \begin{cases} 1, & \text{if task } i \in S \text{ is assigned to machine } m \in M \\ 0, & \text{otherwise} \end{cases}$$

Then the variables for the wireless medium used for data offloading are defined as binary variable w_i, c_i and b_i. The variables represent whether the data is offloaded via WiFi, cellular network, or Bluetooth respectively. The variable that represents the utilized wireless medium for task i is set to 1, otherwise it is set to 0. As an example, the definition of w_i is given below.

$$w_i = \begin{cases} 1, & \text{if task } i \text{ is offloaded via WiFi} \\ 0, & \text{otherwise} \end{cases}$$

In order to simplify the calculation of task completion time, binary variable $o_{i,j,k}$ is defined to represent the order of tasks scheduled on each machine.

$$o_{i,j,m} = \begin{cases} 1, & \text{if task } i \text{ is scheduled before task } j \text{ on machine } m \\ 0, & \text{otherwise} \end{cases}$$

The MIP formulation is given as follows.

$$\text{Minimize: } \max\{t_{i,m}^{finish}\}, \forall i \in S, \forall m \in M \tag{4.1}$$

$$\text{Subject to: } \forall i \in S, \sum_{m \in M} x_{i,m} = 1 \tag{4.2}$$

$$w_i + c_i + b_i + x_{i,src} = 1 \tag{4.3}$$

$$\overline{T}(o_{i,j,m} - 1) \leqslant t_{j,m}^{finish} - t_{i,m}^{start} \leqslant \overline{T} * o_{i,j,m} \tag{4.4}$$

$$x_{i,m} + x_{j,m} + o_{i,j,m} + o_{j,i,m} \leqslant 3 \tag{4.5}$$

$$o_{i,j,m} + o_{j,i,m} \leqslant 1 \tag{4.6}$$

$$t_{i,m}^{start} \geqslant t_i^{arrive} \tag{4.7}$$

$$t_{i,m}^{start} \geqslant t_m^{join} + \overline{T}(x_{i,m} - 1) \tag{4.8}$$

$$t_{i,m}^{start} \geqslant t_{j,m}^{finish} - \overline{T}(1 - o_{j,i,m}) - \overline{T}(2 - x_{i,m} - x_{j,m}) \tag{4.9}$$

$$t_{i,m}^{finish} \leqslant t_m^{leave} \tag{4.10}$$

$$t_{i,m}^{finish} = t_{i,m}^{start} + x_{i,m} * (T_{i,m}^{exec} + T_{i,m}^{trans}) \tag{4.11}$$

$$t_{i,m}^{trans} = w_i \cdot T_{i,WiFi}^{trans} \cdot I_{wifi}^m + c_i \cdot T_{i,3g}^{trans} \cdot I_{3g}^m + b_i \cdot T_{i,bl}^{trans} \cdot I_{bl}^m) \tag{4.12}$$

$$T_{i,src}^{exec} \geqslant x_{i,m} * (T_{i,m}^{exec} + T_{i,m}^{trans}) \tag{4.13}$$

$$\sum_{i \in S} x_{i,m} \cdot E_{i,m}^{exec} \leqslant E_m^{total} \cdot \theta_m \tag{4.14}$$

$$\nu_{src} \cdot E_{i,src}^{exec} > E_i^{trans} \tag{4.15}$$

$$\sum_{i \in S} c_i \cdot C_i^{trans} + \sum_{m \in M} x_{i,m} \cdot C_i^{VM} \leqslant C_m^{budget} \tag{4.16}$$

$$\forall i, j \in S, \forall m \in M \tag{4.17}$$

The objective function minimizes the maximum of the task completion time from all the machines. Constraints (4.2)-(4.12) ensure a valid schedule of the tasks. Constraint (4.2) ensures that each task needs to be assigned to one and only one machine (either local execution or offloading to another machine). Constraint (4.3) ensures that for each task being scheduled, either it is executed locally ($x_{i,src}$) or offloaded via one and only one of the wireless network (w_i, c_i, b_i). Different tasks are scheduled with one of the different type of wireless networks and it can be different for each task based on its execution conditions. $x_{i,src}$ is one of the binary variables $x_{i,m}$.

Constraint (4.4) assigns values to the order variable $o_{i,j,m}$ for each pair of tasks scheduled on the same machine based on the difference between the finish and start time of each task. $\overline{T}$ is the constant greater than the worst case makespan. Constraint (4.5) and (4.6) restrict that the tasks scheduled on the same machine from overlapping, that is, each

machine can only run one task at a time. Since the tasks are generated randomly at times, constraint (4.7) ensures a valid schedule of tasks starting after the task arriving time.

Additionally, the starting and finishing time of a task being scheduled to another mobile device should be within the range of that device's joining and leaving time. This is guaranteed by constraint (4.8) and (4.10). If the task i is not assigned to machine m, $t_{i,m}^{start}$ will be set to a value greater than the worst case makespan by constraint (4.8). Constraint (4.9) indicates that the task is scheduled to an available time slot of the machine. Constraint (4.10) calculates the finish of task i on machine m. The finish time of task i on machine m is equal to the sum of its start time $t_{i,m}^{start}$, the execution time $T_{i,m}^{exec}$ and the data transferring time $T_{i,m}^{trans}$ if $x_{i,m}$ equals to 1.

Constraint (4.13)-(4.16) describe the unique constraints of code offloading in proposed heterogeneous mobile clouds. Constraint (4.13) specifies that tasks that are offloaded should have a shorter execution time than that of local execution. $I_{channel}^{m}$ is a binary indicator that represents the availability to access machine m via each type of wireless interfaces. It is set by the machine when it is initially connected to the network. In the system model, each mobile device is only contributing θ_m proportion of its battery energy for running the offloaded tasks from other mobile device users. This is enforced by constraint (4.14) for each mobile device. When a task is scheduled to offload to another machine, the energy consumption of the data communication should be less than that of executing the task locally. The energy consumption constraint is described in constraint (4.15). The left part of the inequation represents the energy consumption of executing the task i on local processor, and the right side is the energy of computing it on machine m. v_{src} is the energy consumption rate of the original machine of of task i. Finally, constraint (4.16) guarantees that the monetary cost of each machine by using the public cloud services and mobile data does not exceed its limit C_m^{budget}.

We can observe that the formulation is a non-linear model since constraint (4.1), (4.11), (4.13). Generally the mixed integer non-linear programming model is not solvable with the optimization software. Therefore, the model needs to be linearized.

The objective function is a min-max function that can be linearized by minimizing a continuous variable T and adding the following constraint:

$$T \geq t_{i,m}^{finish}, \forall i \in S, \forall m \in M. \tag{4.18}$$

To linearize constraint (4.11), (4.13), three new binary variables $\delta_{i,m}^{medium}$ are defined as follows.

$$\delta_{i,m}^{medium} = x_{i,m} \cdot medium_i, \tag{4.19}$$

where $medium_i$ is w_i, c_i or b_i. All the quadratic terms in constraint(4.11), (4.13) can be equivalently replaced by $\delta_{i,m}^{medium}$, and three constraints are added:

$$\delta_{i,m}^{medium} \leqslant x_{i,m} \tag{4.20}$$

$$\delta_{i,m}^{medium} \leqslant medium_i \tag{4.21}$$

$$\delta_{i,m}^{medium} \geqslant x_{i,m} + medium_i - 1 \tag{4.22}$$

Therefore the mixed integer linear programming formulation is given as:

$$\text{Min} : T \tag{4.23}$$

$$\text{s.t.} : (4.2) - (4.18), (4.20) - (4.22).$$

4.3.2 Complexity Analysis

MCOSP is based on task scheduling problem for heterogeneous computing [220]. Given a heterogeneous computing environment that has processors with different processing speed, and a set of tasks modelled by a directed acyclic graph waiting to be scheduled onto the machines, the objective of the heterogeneous task scheduling problem is to minimize the maximum task completion time while the schedule satisfies the task precedence. The task scheduling in heterogeneous systems is proven to be an NP-hard problem [89], [99]. The main differences of MCOSP are that it considers a set of independent, non-preemptive tasks. Moreover, our proposed problem considers the unique constraints of the shared computing resources to offload in HMC in terms of computing capacity, battery limits, the availability of mobile devices, and network conditions. Therefore, based on the proposed MILP formulation, the MCOSP is an NP-hard problem.

Generally, an MILP problem is solved by Branch-and-Bound method [8]. As a result, there exists a large search space when being solved by branch and bound method due to the number of variables, which is related to the size of the task set and machines. Hence, the optimal results can only be devised for a small set of problem instances. The efficiency of the proposed MILP solution is evaluated in the experiments (Section 4.5). Therefore, an online, lightweight scheduling algorithm for MCOSP is of interest.

4.4 Online Code Offloading and Scheduling Algorithm

In our proposed heterogeneous mobile cloud environment, the central scheduler has no knowledge of future task arrivals. Hence, the scheduler has to make decisions of task offloading and scheduling at runtime without knowing the entire input sequence (i.e. task arrival times). An online optimization framework is therefore needed to solve the MCOSP in real time. We proposed the OCOS algorithm based on the *rent/buy* problem to tackle this challenge.

4.4.1 Mobile Code Offloading and Scheduling Problem

Many online problems involve a sub-problem called *rent/buy* problem. One has to decide whether to stay in current state with a certain amount of cost per time unit, or pay some fixed cost to move to another state.

Ski rental[109] is a classic example. Suppose a person is skiing for an unknown number of days. He needs to either buy the equipment or rent from the shop. Renting costs r per day while buying costs B, where $B > r$. The objective is to make renting or buying decision online in order to minimize the total cost on skiing.

A well-known generalization of this classical *rent/buy* problem is TCP acknowledgement problem [109]. The system needs to decide how long a packet waits before sending the acknowledgement. Waiting incurs delay cost while acknowledging incurs some cost that is more than delay cost. It can be beneficial as multiple packets can potentially be acknowledged together.

Consider the MCOSP problem in the online manner. Similar as TCP acknowledgement problem, MCOSP can be considered as a generalization of *rent/buy* problems. Multiple jobs can be submitted to the scheduler at the same point of time. Hence as the service continues, there will be several mobile tasks waiting on the scheduler to be dispatched. In particular, consider the problem as to whether to hold a task waiting in the scheduler or offload to one of the machines in HMC to execute. The tasks may finish execution earlier if they wait in the scheduler for some time and execute on another machine that is available later other than the current available ones. A task waiting to be processed sometime in the future generates a waiting cost of 1 for each time unit. Otherwise, it can

pay an additional cost B to execute on one of the machines immediately. Obviously, if the scheduler somehow knows the task needs to wait for at least B time periods, it is optimal to pay a higher cost and run the task immediately in one of the machines at the beginning. However, in reality the scheduler has no knowledge of that. Therefore, for each task in the queue, the scheduler needs to decide at each time period whether to keep the task waiting or offload to a machine considering the scheduler does not have any knowledge of the next task arrival. We define the following entities in the context of MCOSP:

1. **Rent:** The scheduler holds the task awaiting in the queue for one time period.

2. **Buy:** The scheduler dispatches the task to one of the machines for execution.

3. **Renting cost R_i:** Renting incurs one cost per time unit. Thus, the renting cost R_i of task i refers to the time period task i waited before being dispatched. R_i is defined as:

$$R_i = T_i^{wait}$$

4. **Buying cost $B_{i,m}$:** Buying cost refers to the time consumed for the task to finish execution. Assume task i is assigned to machine m for execution. $B_{i,m}$ is defined as:

$$B_{i,m} = T_m^{avail} + T_{i,m}^{exec} + T_{i,m}^{trans} - T_i^{arrival},$$

where T_m^{avail} is the earliest available time of machine m, $T_{i,m}^{exec}$ is the execution time of task i on machine m, $T_i^{arrival}$ is the arriving time of task i. To calculate task execution time of task i on machine m, if there is an task offloading required, the data transmission time for offloading is calculated based on the wireless channel availability of machine m (i.e. I_{wifi}^m, I_{3g}^m, I_{bt}^m) in Eqn.(4.12). If multiple channels are available, then the channel with minimum transmission time is selected. The leaving time of a machine is considered when calculating the cost. That is, if the leaving time of a machine m is before a task i can finish computation, $T_{i,m}^{exec}$ is set to infinity so that the overall buying cost would be infinity.

4.4.2 Online Code Offloading and Scheduling Algorithm

Based on the problem definition above, we proposed an online code offloading and scheduling algorithm (OCOS) based on break-even algorithm. The break-even algorithm

Algorithm 3 Online Code Offloading and Scheduling Algorithm

1: *Initialize* : *waitlist* $L, n \leftarrow 0$
2: **for** $\forall m \in M$ **do**
3: initialize machine properties $\mu_m, r_m, \theta_m, \nu_m, PR^{active}, PR^{idle}$
4: **while** $n \leqslant D$ **do**
5: Upon receiving a task i: $L \leftarrow i$
6: $R_i \leftarrow 0$
7: **for** $\forall m \in M$ **do**
8: update T^m_{avail}
9: $B_{i,m} \leftarrow 0$
10: **for** $\forall i \in L$ **do**
11: **for** *machine* $m \in M$ **do**
12: calculate $B_{i,m}$
13: $B_{i,m*} \leftarrow min_{m \in M} B_{i,m}$
14: **for** $\forall i \in L$ **do**
15: $R_{min} \leftarrow$ MAX_VALUE
16: **if** $R_i \geqslant B_{i,m*}$ and $R_i < R_{min}$ **then**
17: $R_{min} \leftarrow R_i$
18: $tag \leftarrow i$
19: **if** $C_{tag,m^*} + C^{current}_{m^*} \leqslant C^{budget}_{m^*}$ and $E_{tag,m^*} + E^{current}_{m^*} \leqslant E^{total}_{m^*} \cdot \theta_{m^*}$ **then**
20: Schedule task tag to machine m^*
21: Remove task tag from L
22: **for** $\forall i \in L$ **do**
23: $R_i \leftarrow R_i + 1$
24: Update R_i
25: $n \leftarrow n + 1$

[110] has been widely used to design the online algorithms. A break-even point refers to the time when the renting cost equals to the buying cost. The algorithm decides to buy the resources after the break-even point. It has been proven to be 2-competitive [110].

A discrete time horizon of D epochs is considered, where D could possibly be infinite. Starting from time 0, the tasks arrive at the scheduler at arbitrary epoch. The pseudo code of the proposed algorithm is given in Algorithm 3. Firstly, the scheduler initializes a waitlist L and updates all the information of the machines available (e.g. earliest available time, processing speed, energy rate, etc.) by sending a request to each machine (step 1-3). If there is any change of the information, it will be sent to the scheduler periodically. When a task is generated, its information is sent to scheduler and is added into the waitlist and its renting cost is set to 0 (step 5-6). At the beginning of each scheduling epoch, the scheduler updates the earliest available time of each machine (step 10-12). The

machine m^* that has the minimum buying cost is selected (step 13). Then it calculates the renting cost and buying cost on every machine for each task in the waiting list. Tasks meeting the following condition will be selected and scheduled to the corresponding machine m with the least buying cost $B_{i,m}$:

$$R_i \geqslant \min_{m \in M} B_{i,m}. \tag{4.24}$$

If there are more than one task meeting the above-mentioned condition for the same machine, the task with the lowest R will be scheduled (step 14-18). Note that to follow the energy and monetary cost constraints, the algorithm maintains two variables, $C_m^{current}$ for current total monetary cost of machine m (i.e., mobile user m) and $E_m^{current}$ for current overall energy consumption for machine m, to reject the schedule if the conditions are not met (step 19-21). Scheduled tasks will be removed from the waiting list. Then the scheduler updates the renting cost of tasks in the waitlist by adding one time unit (step 29-32).

The complexity of the proposed algorithm is $O(|M| \cdot |S|)$, given that $|S|$ tasks arrived during the time period D and there are $|M|$ machines available. Step 6 takes $O(|M|)$ to update the available time at each epoch. Updating buy cost for each task in the waiting list takes $O(|M| \cdot |S|)$ (step 6-10). Step 11 takes $O(|S|log|M|)$ to check each task's condition. Therefore the overall algorithm takes $O(|M| \cdot |S| + |M| + |S|log|M|)$ at each epoch, which can be summed as $O(|M| \cdot |S|)$. We then present a **competitive analysis** on the online algorithm as follows.

Definition 4.1 (Competitive Ratio)**.** *An online algorithm ALG is **c-competitive** if for all finite input sequences I,*

$$ALG(I) \leq c \cdot OPT(I) + \alpha$$

where $ALG(I)$ is the cost of the online algorithm and $OPT(I)$ is the cost of the offline optimal [20]. A c-competitive online algorithm ALG is also a c-approximation algorithm with the restriction that ALG must compute online.

The competitive ratio is evaluated to show the performance of the proposed online algorithm by calculating the total completion time of the tasks in a time period for the offline optimal solution and the online algorithm.

Let $T_{\sigma,m}^{ALG}$ and $T_{\sigma,m}^{OPT}$ denote the makespan of tasks scheduled on machine m by the proposed OCOS algorithm and offline optimal respectively, where σ denotes the set of

tasks scheduled to machine m by OCOS or offline optimal. Given a heterogeneous set of machines M, and an arbitrary task sequence σ to schedule among machines in M, we obtain the following:

Lemma 4.1. *For $\forall m \in M$, $T^{ALG}_{\sigma_A,m} \leqslant 2\ T^{OPT}_{\sigma_O,m}$, where $\sigma_A \in \sigma, \sigma_O \in \sigma$ are the task sequences scheduled to machine m by OCOS algorithm and offline optimal algorithm respectively.*

Proof. For task set σ_A and σ_O, there are two cases to discuss where $\sigma_A = \sigma_O$ and $\sigma_A \neq \sigma_O$.

First, if $\sigma_A = \sigma_O$, based on the scheduling policy of OCOS algorithm (Equation 4.24),

$$T^{ALG}_{\sigma_A,m} = \sum_{i \in \sigma_A} (R_i + B_{i,m}) = 2 \sum_{i \in \sigma_A} B_{i,m}$$

$$T^{OPT}_{\sigma_O,m} = \sum_{i \in \sigma_O} B_{i,m} = \sum_{i \in \sigma_A} B_{i,m} = \frac{1}{2}\ T^{ALG}_{\sigma_A,m}.$$

Thus, for $\sigma_A = \sigma_O$, $T^{ALG}_{\sigma_A,m} = 2\ T^{OPT}_{\sigma_O,m}$

Second, if $\sigma_A \neq \sigma_O$, the possible relations between σ_A and σ_O are either $\sigma_A \subset \sigma_O$ or $\sigma_A \not\subset \sigma_O$. We proof by contradiction that the case of $\sigma_A \not\subset \sigma_O$ does not exist.

Assume $\sigma_A \not\subset \sigma_O$ exists, then let $\sigma' = \sigma_A - \sigma_O$ that σ' is scheduled to machine m by the OCOS algorithm. Based on OCOS algorithm's scheduling policy that dispatches task i only when the renting cost R_i is equal to the minimum of buying cost $B_{i,m}$ among M (Equation 4.24), $\sum_{i \in \sigma'} B_{i,m}$ is the minimum. On the other hand, since σ' is scheduled to another machine by the offline optimal algorithm, for instance, machine n, we have $\sum_{i \in \sigma'} B_{i,n} \leqslant \sum_{i \in \sigma'} B_{i,m}$, which is contradicted to the case which $\sum_{i \in \sigma'} B_{i,m}$ is the minimum. Therefore, $\sigma_A \not\subset \sigma_O$ does not exist.

When $\sigma_A \subset \sigma_O$,

$$\frac{T^{ALG}_{\sigma_A,m}}{T^{OPT}_{\sigma_O,m}} = \frac{T^{ALG}_{\sigma_A,m}}{T^{OPT}_{\sigma_A,m} + T^{OPT}_{(\sigma_O - \sigma_A),m}} \leqslant \frac{T^{ALG}_{\sigma_A,m}}{T^{OPT}_{\sigma_A,m}} = 2.$$

Therefore, for $\forall m \in M$, $T^{ALG}_{\sigma_A,m} \leqslant 2\ T^{OPT}_{\sigma_O,m}$. This completes the proof. □

Theorem 4.1. *The proposed OCOS algorithm for task scheduling on heterogeneous mobile cloud environment is $2-competitive$.*

Proof. Let ALG denote the proposed OCOS algorithm and OPT denote the offline optimal algorithm. Let $\sigma = (t_1, t_2, ..., t_n)$ be an arbitrary tasks sequence submitted for scheduling, and $\forall m \in M$ be a machine in the environment to execute tasks. $ALG(\sigma)$ and $OPT(\sigma)$ denote the makespan of the schedule generated by OCOS algorithm and the offline optimal schedule respectively on the input sequence σ.

Without loss of generality, we assume that $\sigma = \sigma_1 \cup \sigma_2 \cup ... \cup \sigma_m$ is the task sequence scheduled on each machine by algorithm OPT, and

$$OPT(\sigma_1) \leqslant OPT(\sigma_2) \leqslant ... \leqslant OPT(\sigma_m),$$

Let $\sigma = \sigma'_1 \cup \sigma'_2 \cup ... \cup \sigma'_m$ be the task sequence scheduled by algorithm ALG and

$$ALG(\sigma'_1) \leqslant ALG(\sigma'_2) \leqslant ... \leqslant ALG(\sigma'_m),$$

where σ_m and σ'_m are tasks scheduled to machine m by OPT and ALG respectively. Thus, the makespan $OPT(\sigma) = OPT(\sigma_M)$ and $ALG(\sigma) = ALG(\sigma'_m)$.

Based on the definition of competitive ratio and Lemma 1,

$$\frac{ALG(\sigma)}{OPT(\sigma)} = \frac{ALG(\sigma'_m)}{OPT(\sigma_m)} \leqslant 2.$$

Hence, the competitive ratio of OCOS algorithm is 2. □

4.5 Performance Evaluation

In this section, we evaluate the proposed algorithms in two aspects, namely the scheduling performance (i.e., the makespan of the tasks) and the offloading performance (i.e., total execution time and energy saved). Six sets of experiments are conducted to evaluate the proposed offline and online scheduling solutions in terms of scheduling performances, algorithm efficiency and the effect of the parameters considered by the algorithms. For the scheduling performance, the proposed algorithms are compared with the online mode independent task scheduling heuristic opportunistic load balancing (OLB) [24]. For the offloading performance, the proposed algorithms are compared with the offloading policies in ThinkAir [122].

4.5.1 Experiment Settings

For the offline mixed integer linear programming model proposed, the offline optimal algorithm is implemented in Gurobi Optimizer 6.5 [168] to provide the offline optimal solutions of the model. The optimization is undertaken on an m1.xlarge instance on Nectar Cloud [165] with 8 vCPUs and 32 GB RAM. For the online algorithm evaluation, we develop our own experiment environment to evaluate the performance of the proposed algorithms. The scheduler runs on a local computer that has a Intel Core i7 CPU with 3.4 GHz and 8GB RAM. Experiment settings and workload parameters are listed in Table 4.3.

Table 4.3: Parameter settings for evaluation

Parameter	Value
Task data size (f)	[0.05,0.5],[1,5]
Task computation (ϖ)	[1,10],[50,100]
Mobile device CPU Frequency (μ)	{0.2,0.5,0.8,1.0,1.2}
Cloudlet CPU Frequency	2.5
Cloud VM CPU Frequency	3.6
Cloud VM charge rate	0.84
Battery limit fraction (θ)	Uniform(0.2,0.5)
Active CPU power consumption rate (PR^{active})	0.07,0.34,0.48,0.56,0.6
Idle CPU power consumption rate	0.002
WiFi data rate (BW_{wifi})	1
Bluetooth data rate (BW_{bt})	0.26
Cellular data rate (BW_{cell})	0.85
WiFi power rate (ρ_{wifi})	1.94
Bluetooth power rate (ρ_{bt})	0.28
Cellular power rate (ρ_{cell})	5.56
Weibull parameters (α, β)	(1.9543,326.87),(1.2861,178.56), (0.8712,276.87), (1,163)

Table 4.4: Characteristics of workloads

Type	Data Size (MB)	CPU Cycle (Giga)
LCSD	[0.05, 0.5]	[1, 10]
HCSD	[1, 5]	[50, 100]
HCLD	[1, 5]	[50, 100]

Due to the lack of real-world traces that are suitable for HMC environment and mobile device constraints, workloads are obtained by profiling the cognitive applications running on mobile devices in the experimental environment. The workload consists of three types of task sets that represent the diversity of mobile application tasks, namely low computation small data size (LCSD), high computation small data size (HCSD), and high computation large data size (HCLD). In order to profile values of the parameters of task sets, an open-source OCR application[2] is executed on Android with Android Studio performance analysis tools to profile the OCR application on the method level. The profile, which consists of data size of the captured picture frame and inclusive CPU running time of each operation (i.e. methods for Android application) for processing, is then classified into the three workload types. In workload LCSD, the range of data size of tasks lies in the interval [0.05, 0.5] megabytes, and the computation of completing each task distributes in the interval [1, 10] giga CPU cycles. Similarly in workload HCSD and

[2]Available at https://github.com/rmtheis/android-ocr

HCLD, data size of the tasks are in the interval of [0.05, 0.5] and [1, 5] megabytes respectively, and the computation of both task set is in a interval of [50, 100] giga CPU cycles. A summary of the workloads used in the experiment is listed in Table 4.4.

Regarding the experiment environment, it includes 2 identical cloudlets and 3 identical VMs in the public cloud. The number of mobile devices varies based on different experiments conducted. The hardware information is profiled from a number of Android smartphones. For each mobile device, the CPU speed μ is randomly assigned from the set $\{0.2, 0.5, 0.8, 1.0, 1.2\}$ GHz for the mobile devices. These CPU frequencies are obtained from different mobile devices such as Nexus 4, HTC G13, HTC G3, and Samsung I997. The CPU speed of cloudlet is 2.5 GHz, and cloud VM CPU speed is 3.6 GHz following the Amazon EC2 C3 instances. The monetary cost of cloud VM is \$0.84 per offloading request. θ_i is uniformly selected from the interval [0.2, 0.5]. For the energy aspect of the mobile devices, the parameters given in the energy models proposed by Ali et al. [6] are adopted in the experiments. The active CPU energy consumption rate PR^{active} is set from $\{0.07,0.34,0.48,0.56,0.6\}$ based on the above-mentioned CPU frequency accordingly. The idle power consumption rate PR_{idle} is set to 0.002 for all devices.

The network of the experiment environment consists of three types of wireless mediums, namely mobile cellular network, Bluetooth, and WiFi. Network parameters are obtained by profiling the network conditions in our experiment environment using file transferring applications on the mobile device. WiFi network speed BW_{wifi} is set to 1 MBps, Bluetooth transmission speed BW_{bl} is set to 0.26 MBps, and cellular network speed BW_{cell} is set to 0.85 MBps. The network latency to public cloud service is set to 0.1s. It is profiled by testing the delay to the Amazon EC2 service in Sydney region. Moreover, the energy consumption parameters of WiFi, Bluetooth, and cellular network are set to $\rho_{wifi} = 1.94\ W$, $\rho_{bt} = 0.28\ W$ and $\rho_{cell} = 5.56\ W$ respectively based on the energy model proposed by Balasubramanian et al. [15]. A 2-parameter Weibull distribution is used to obtain the leaving time of the mobile device from HMC. To capture the mobility patterns of real mobile devices in the wireless network, the CRAWDAD tracesets [123] are used to obtain valid values for the shape parameter α and the slope parameter β in the Weibull distribution. The traceset is composed of the system logs of WiFi access points on Dartmouth campus from September 1, 2005 to October 4, 2006. The number of sessions in

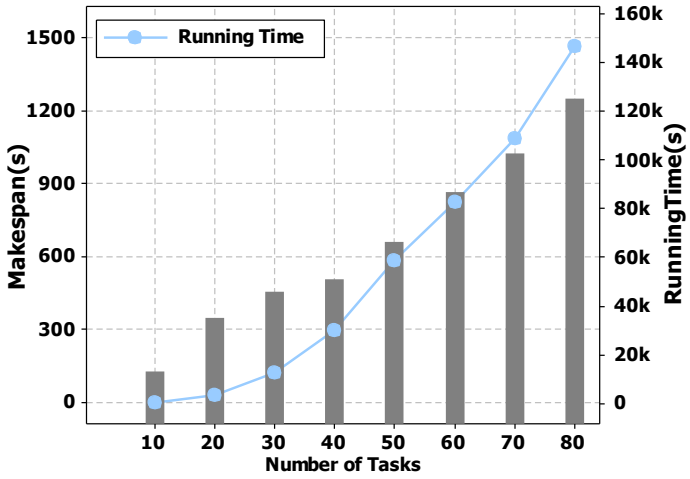


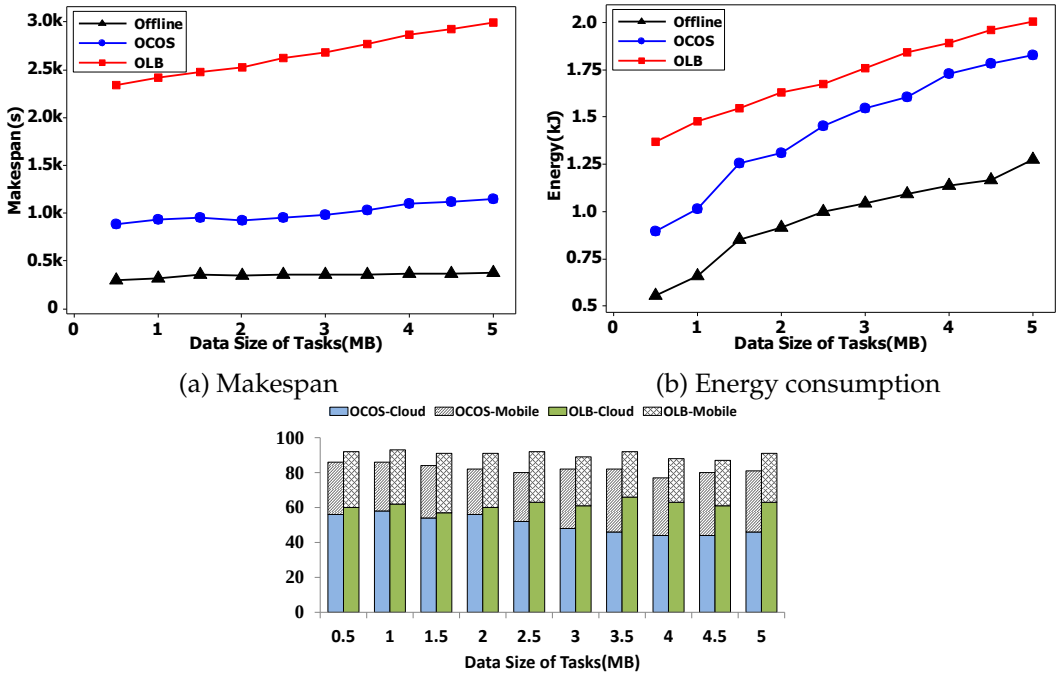
Figure 4.1: Makespan and running time of the offline optimal approach

every minute is extracted from the trace to generate the probability function using rank regression [191]. Then the time-to-failure of each machine is randomly obtained from the inverse function of Weibull distribution in order to generate the t^{leave}.

4.5.2 Experiment Results

Offline Approach Performance Evaluation

Figure 4.1 shows the time taken to generate the solution and the makespan of tasks in the workload by using the offline approach. The task set for the experiment is obtained from workload HCLD. Note that, since the offline approach uses Branch-and-Bound (BB) algorithm to solve the MILP model formulated, the search space of BB algorithm is only based on the number of tasks and machines rather than the heterogeneity of the tasks and machines. Hence, either workload can be used for the evaluation of the offline approach. Results of makespan for the offline optimal are used as a benchmark for the online algorithms evaluation. As shown in Figure 4.1, the processing time for generating the optimal solution increases expeditiously as the number of tasks grows. This is because the complexity of our proposed MILP model is a product of the number of tasks, machines, and the variables defined. Hence, solving an offline optimal schedule that involves large data sets would require an unreasonably large amount of time for the mobile cloud offloading systems, which demands lightweight and timely offloading decisions. Therefore, the



(a) Makespan

(b) Energy consumption

(c) Percentage of tasks offloaded to cloud and peer mobile devices

Figure 4.2: Performance of OCOS algorithm with different task data size

results show necessity of the online scheduling algorithm to solve the proposed mobile cloud offloading and scheduling problem with low processing delay.

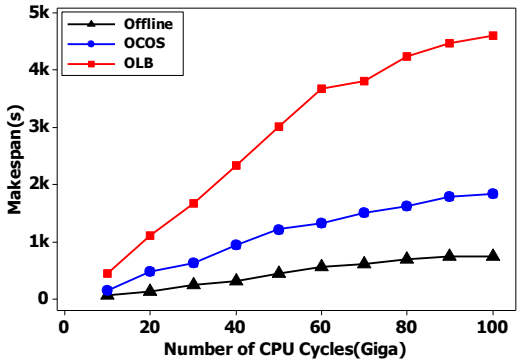
OCOS algorithm Performance Evaluation

The performance of the proposed OCOS algorithm is investigated. The online solutions under all three types of workloads are evaluated and compared with the non-preemptive task scheduling heuristic OLB as well as the benchmark offline optimal schedule. The OLB heuristic schedules tasks to the machine with the earliest available time when tasks arrive at the scheduler. Note that, OLB heuristic is different from OCOS algorithm that it does not consider the offloading benefits (e.g. whether the offloading would shorten the task completion time) when scheduling the tasks. Two metrics are considered, namely makespan and the overall energy consumption. The makespan represents the maximum task completion time of the set of scheduled tasks. The overall energy consumption is the sum of each mobile device's energy consumption for running the scheduled tasks.

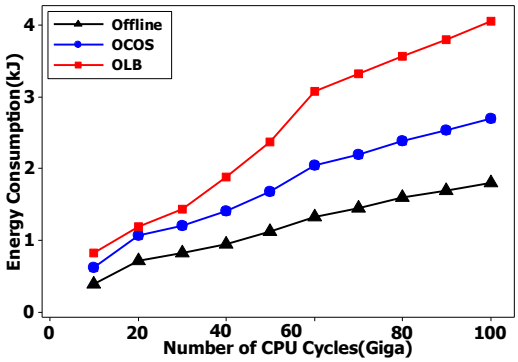
To compare the impact of different offloading data sizes on the performance of OCOS

algorithm, tasks from workload HCSD and HCLD are categorized into 10 task sets based on the offloading data size (from 0.5 MB to 5 MB), CPU cycles from 50 to 60 giga cycles. Results are averaged by 50 runs. Figure **??** depicts that the makespan of OCOS algorithm and offline approach algorithm only have a small increase as the offloading data size of the tasks increases. OLB heuristic generates much higher makespan. This is caused by its scheduling strategy that only considers earliest available time of the machines, which makes the load of machines unbalanced. Also, as the data size increases, there is a much faster growth in makespan than OCOS algorithm and offline approach since mobile devices take longer time to transfer the offloaded data. The results indicate that, for the cognitive application with offloading data size less than 5MB, the performance of OCOS is not affected by the data size. It can also be observed makespans generated by the online algorithm are around 2.6 times as much as the offline approach on average, while OLB heuristic is 5 times longer. Different from the 2x makespan shown in Theorem 5.3, the 2.6x makespan yielded by the experiment is caused by the difference of modelling and hardware in machines. The models for task execution time have a high level of hardware abstraction to eliminate the complexity lying in hardware (which is not the focus of this work). However, in reality machines have multi-core CPU and multi-level memory hierarchy I/O that will affect the task execution time. Moreover, in reality, mobile devices and HPC servers run many background activities that will affect and lengthen the execution time of tasks in the HMC environment.

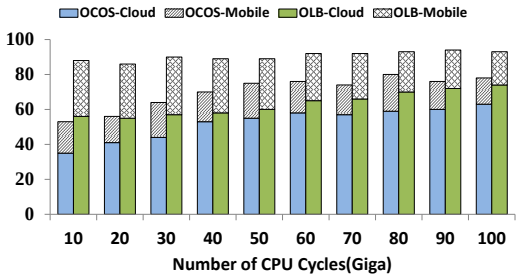
The overall energy consumption of the mobile devices in the experiments (Figure 4.2b) increases as the data size grows. For OLB heuristic, the percentage of tasks scheduled to cloud and peer mobile devices does not have much fluctuation (shown in Figure 4.2c) with the growth of data size, due to the earliest-available-time-only scheduling strategy. The increase in the energy consumption of the scheduled made by the OLB heuristic is caused by the larger data size and the higher makespan as the data size increases. For OCOS algorithm, more tasks are scheduled to peer mobile devices (shown in Figure 4.2c) other than cloud as the data size of tasks grows. This is because OCOS considers the task completion time as well as the energy consumption of executing tasks. Tasks with larger data size tend to be offloaded to peer mobile devices rather than cloud to reduce the data offloading energy consumption as well as the transferring time and generate balanced



(a) Makespan with different WiFi bandwidth usages



(b) Average energy consumption per machine with different WiFi bandwidth usages



(c) Makespan with different Weibull Distribution (α: shape, β: scale)

Figure 4.3: Performance of OCOS algorithm for tasks with different computing requirements

schedules. As a result, the overall energy consumption of all mobile devices increases. In addition, the growth in data size also increases the energy consumption of the wireless transmitters.

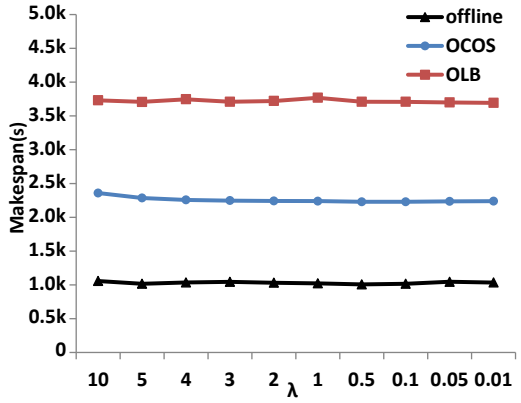
Furthermore, for tasks with different computing resource requirements, the impact on the performance of the proposed algorithms is evaluated. For each run, the tasks from workload type LCSD and HCSD are categorized into the 10 task sets based on the CPU cycles required (10 to 100). Results are averaged by 50 runs. The makespan of schedules, total energy consumption and portion of tasks offloaded to cloud as well as peer mobile devices are shown in Figure 4.3. In Figure 4.3a, the makespans generated by all three algorithms increase with the increase in computation load. The makespan generated by OLB heuristic grows much faster than that of OCOS algorithm due to its offloading strategy that tasks with high computation requirements are scheduled to low computing capacity mobile devices. Eventually, the makespan of the unbalanced sched-

ule becomes higher. On the other hand, the growth of makespan generated by OCOS algorithm is much slower than that of the OLB heuristic. This is because the OCOS algorithm compares the current renting cost (waiting time) of the task with its buying cost (task completion time) on different machines when making the scheduling decisions, and thus generates a more load balanced schedule comparing to OLB. In terms of energy consumption, Figure 4.3b shows that the differences of growth trend between three algorithms are similar to those of the makespan in Figure 4.3a. This is because the longer the execution time of the task is, the higher energy mobile devices consume. Therefore, as the computation requirements of the tasks grow, the overall energy consumption of the mobile devices increases. In average, the energy consumed executing the tasks of the schedule generated by OCOS algorithm is around 2 times less than that of OLB, and around 1.5 times more than that of the offline approach.

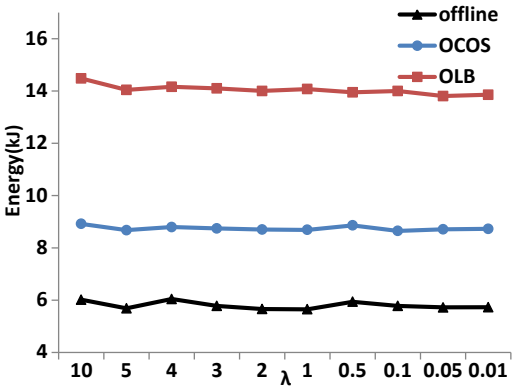
Figure 4.3c shows the percentage of tasks scheduled to offload to cloud and peer mobile devices by OCOS algorithm and OLB heuristic respectively under different computing requirements. For OCOS algorithm, the portion of tasks offloaded to cloud and the total portion of tasks offloaded both increase with the increase of computing requirements of tasks. This is because OCOS considers the task completion time as well as how much time and energy can be saved when deciding the schedule. Therefore, as the computing requirements of the tasks increase, OCOS schedules more tasks to cloud to reduce the makespan. On the contrary, OLB heuristic only considers the earliest available time of the machine when scheduling tasks. Thus, most of the tasks (90 percent on average) are offloaded to cloud and other mobile devices regardless of the offloading benefits. This scheduling strategy also causes unbalance schedules among tasks (shown in Figure 4.3a), comparing to OCOS that takes machine load balance into consideration.

Based on the results obtained from the two sets of experiments conducted above, compared to OLB, OCOS algorithm has a steady performance on makespan and energy consumption for cognitive applications with task data size between 1MB and 5MB. The performance ratio of the online algorithm over offline approach is around 2 for makespan.

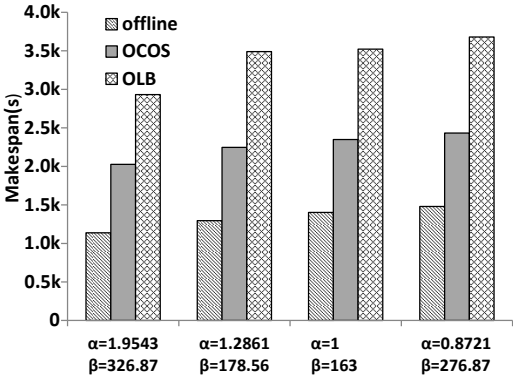
To observe the performance of algorithm in terms of mobile device reliability, two sets of experiments are conducted. The first set compares the makespan and energy consumption of the three algorithms with different task arriving rates (i.e. λ). The second



(a) Makespan with different WiFi bandwidth usages



(b) Average energy consumption per machine with different WiFi bandwidth usages



(c) Makespan with different Weibull Distribution

Figure 4.4: Makespan with different Weibull Distribution (α: shape, β: scale)

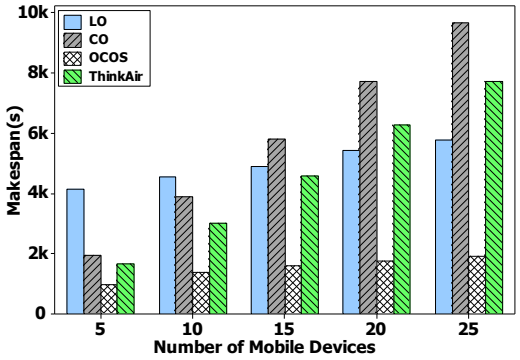
experiment tests the effect of different mobile device reliability (i.e. device leaving time) on makespans of scheduled tasks. Tasks from workload HCSD are used for both sets of experiments. Results are shown in Figure 4.4a-4.4c. From Figure 4.4a and Figure 4.4b we can observe that task arriving rate does not have much influence on makespan and overall mobile device energy consumption since the results do not have fluctuations over different arriving rates. This is due to the fact that all three algorithm aim to schedule multiple tasks that match its scheduling policies at the same time. Moreover, makespan and energy consumption generated by OCOS algorithm are around 65% and 57% of OLB algorithm respectively, and 2.2 and 1.5 times of offline algorithm respectively.

To test the performance of proposed algorithms on mobile device reliability, multiple cases are tested to represent different device mobility as device leaving time. The

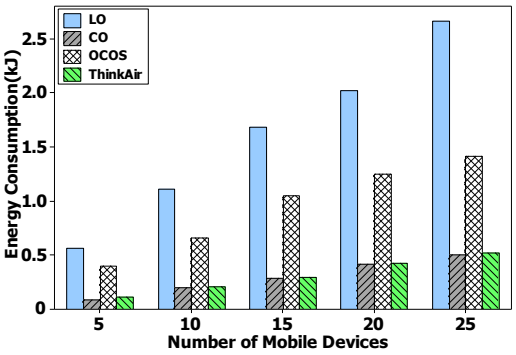
device leaving time are obtained from the above-mentioned Weibull distributions in Section 4.5.1 using CRAWDAD tracesets. The shape (α) and scale (β) parameters for the Weibull distributions are listed in Table 4.3. The same task set from workload HCSD is used for all the cases. α, β for first two cases are extracted from data between 1pm and 8pm in tracesets, the fourth case is obtained from data between 5am and 1pm, and the third case is a synthetic pair of parameters for comparison. $\alpha < 1$ indicates device failure rate decreases with time, that is, mobile devices have short connection time. $\alpha > 1$ indicates device failure rate increases with time and mobile devices have longer connection time, while $\alpha = 1$ represents a constant failure rate. In Figure 4.4c, it shows that as device failure rate increases (i.e., mobile device connection time decreases), the makespan generated by OCOS has an approximate 10% increase throughout four cases, while makespans generated by OLB have 20% increase. This is due to the fact that more tasks are scheduled onto cloud servers instead of mobile devices that left the environment. The makespans of OCOS algorithm are around 37% less of those generated by OLB algorithm since OCOS considers load balancing of machines during scheduling.

Moreover, the performance of OCOS algorithm in terms of offloading benefits (e.g. execution time and energy saved) against conventional offloading strategies is evaluated. Upon comparison, three baselines are implemented. The first baseline, offloading to cloud only (CO), always offloads the submitted mobile tasks to the cloud VMs. The second base online, Mobile Local Only (LO) always executes mobile tasks on its own mobile device locally. The third baseline algorithm is the mobile code offloading strategies proposed in ThinkAir with the execution time priority policy [122]. Note that the offloading decision making algorithm in ThinkAir only considers the context of a single mobile device to cloud offloading model. All algorithms are evaluated with same task sets from workload type HCLD. The size of the task set is 20 tasks generated per mobile device on average. Results are shown in Figure 4.5 with different number of mobile devices.

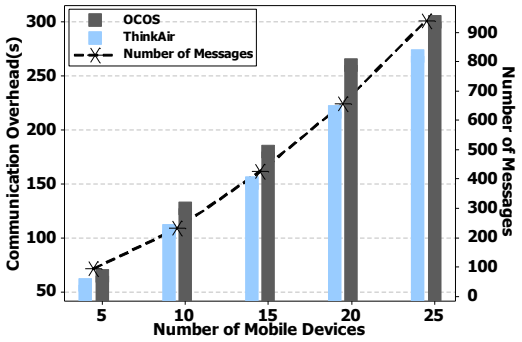
As shown in Figure 4.5a, the makespan generated by all four algorithms increase as the number of mobile device users increases. For baseline LO and CO, the growth of CO is much faster than that of the LO. Similarly, the makespan generated by ThinkAir increases faster than the LO baseline. The results indicate that as the number of mobile device users increases, the benefits of offloading to fixed public cloud resources decreases.



(a) Makespan with different WiFi bandwidth usages



(b) Average energy consumption per machine with different WiFi bandwidth usages



(c) Makespan with different Weibull Distribution (α: shape, β: scale)

Figure 4.5: Performance comparison of different offloading strategies

This is because both baseline algorithms make individual offloading decisions without considering the overall computation load of the cloud resources. As a result, the task offloaded by one mobile device may be delayed in the cloud VM, which ends up with a longer makespan. On the contrary, the makespan generated by OCOS algorithm increases much slower comparing to the other three baselines. This is because OCOS uses the peer mobile devices in HMC environment, and considers the computation load of each machine in the environment when making the offloading decisions. It can also be observed that the ratio of the makespan of ThinkAir over that of the OCOS algorithm increases as the number of users increases. The growth shows that OCOS algorithm outperforms conventional offloading strategies (e.g., CO, LO and ThinkAir) when the number of mobile users is large.

Furthermore, the performance of OCOS algorithm in terms of overall mobile device energy consumption is illustrated in Figure 4.5b. It shows that the overall mobile device

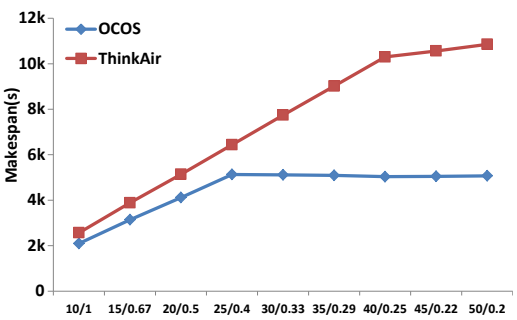
energy consumption in HMC for all the algorithms increases with the growth in number of mobile users. The energy consumption of algorithm CO and ThinkAir is relatively low comparing to the other two algorithms since the mobile devices consume much less energy in the idle mode when offloading tasks to cloud. Compared to that, for OCOS algorithm, the overall energy consumption for all the mobile devices is around 50% of LO on average, and twice as much as that of CO and ThinkAir. This is because the objective of OCOS algorithm is to minimize the makespan with the constraints of mobile devices such as battery lifetime and mobility. Thus, although the energy consumption is higher than CO and ThinkAir, the makespan generated by OCOS algorithm is one fourth and half of the makespan generated by CO and ThinkAir respectively, and can get much lower when the number of mobile users scales up (as shown in Figure 4.5a).

In addition, the communication time overhead for operations related to communication such as data offloading and messages passing are evaluated, and results are shown in Figure 4.5c. The communication overhead of OCOS is slightly larger than that of ThinkAir due to the use of Bluetooth when offloading to a peer mobile device, which incurs a longer communication time comparing to WiFi. The average communication time overhead for OCOS algorithm is around 12% of the makespan. Although it is slightly larger than ThinkAir, the offloading gain (i.e. makespan saved) is much more than ThinkAir. Furthermore, as a centralized scheduling algorithm, it is necessary to measure the behaviour of the number of control messages sent by the scheduler to update the hardware information of the mobile devices and cloud. The results are shown in Figure 4.5c with the dash line. The number of messages sent by the scheduler increases as more mobile users join the environment. Since ThinkAir makes offloading decisions locally on each mobile devices, it has less communication overhead than OCOS. However, the difference of the communication overhead is rather small comparing to the shorter makespan enabled by OCOS.

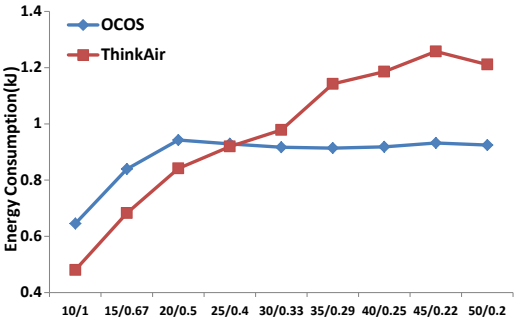
In reality, there are usually limited numbers of WiFi access points available in a mobile cloud network, therefore the limited network bandwidth may affect the performance of the proposed algorithm. A set of experiments are conducted to analyse the influence of network bandwidth usages. The effects of cellular network, WiFi-direct and Bluetooth are not considered here as their bandwidths are not affected by the number of machines

in the network. The experiment is set with one available WiFi access point and it provides the fixed bandwidth of 10 Mbps. All the machines share the network bandwidth evenly. The task set containing a number of 500 tasks obtained from workload type HCLD are used for the experiments. The results are averaged by 50 runs and shown in Figure 4.6a and Figure 4.6b in comparison with results of ThinkAir. In Figure 4.6a, as the average bandwidth per machine decreases from 1 Mbps, the makespan of OCOS increases from around 2000 seconds to 5000 seconds, and becomes stable after 0.4 Mbps since the bandwidth does not change too much after that, while the makespan of ThinkAir increases more rapidly. The increase of makespan is due to the competition of bandwidth between devices leading to longer data transferring time over WiFi to public clouds. Also, OCOS schedules some tasks with large data size to run local instead, which generates longer makespan. The similar results also reflect on the average energy consumption per mobile device. The shorter makespan and less energy consumption generated by OCOS, especially with more devices, is because of the use of mobile device cloud that is able to load balance the computation among the entire network.

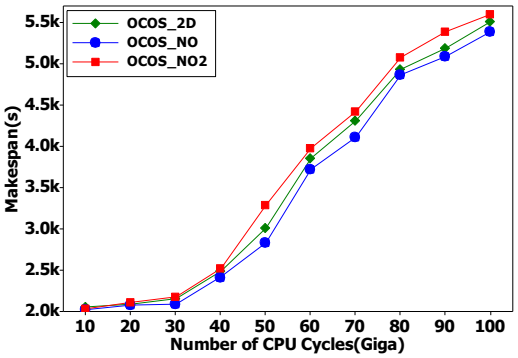
Since machines especially mobile devices may drop out of the network before its estimated leaving time, the robustness of OCOS algorithm is evaluated. The makespans generated over different task computation workload are compared in three cases: 1) no machine dropouts, 2) 2 randomly chosen machines dropping out at random time before its estimated leaving time, and 3) no machine dropouts but without the two machines chosen in case 2 in the network. The results are averaged by 50 runs and depicted in Figure 4.6c. *OCOS_NO* represents case 1, *OCOS_2D* represents case 2, and *OCOS_NO2* represents case 3. As shown in Figure 4.6c, the machine dropouts do not have a significant effect on the makespan generated by OCOS algorithm. Compared with the results of *OCOS_NO*, the makespan has around 0.05% increase for different amount of computation workload with machines accidentally dropping out of the network. This is because the OCOS algorithm receives updates on the earliest available time from each machine in the network at the beginning of each scheduling epoch. If a machine drops out before its estimated leaving time, the scheduler is able to detect it and considers the rest of machines. Moreover, the makespans of OCOS algorithm with machine dropouts (*OCOS_2D*) are lower than those generated by OCOS algorithm with two less machines in the network



(a) Makespan with different WiFi bandwidth usages



(b) Average energy consumption per machine with different WiFi bandwidth usages



(c) Makespan with different Weibull Distribution (α: shape, β: scale

Figure 4.6: Effect of network bandwidth on the scheduling performance

from the beginning (*OCOS_NO2*) due to the load balancing of OCOS algorithm.

4.6 Summary

In this chapter, we discussed the mobile code offloading and scheduling problem (MCOSP) in the heterogeneous mobile cloud environment. We formally defined the problem with a mixed integer programming model and devised optimal offloading and scheduling results by solving the optimization model offline. In order to provide an online scheduling solution for our proposed mobile code offloading and scheduling problem that is near-optimal, we further designed a real-time scheduling algorithm based on the ski-rental framework. We show that the proposed online scheduling algorithm OCOS is $2-$ *competitive* of the offline optimal solution. The experimental results show that the OCOS algorithm is consistent with the offline optimal solution in terms of competitive-

ness. The algorithm generates shorter makespans and less energy consumption with the cognitive applications such as OCR and face detection applications comparing to OLB heuristic, and also scales well comparing to other previous proposed mobile code offloading strategies when the number of mobile device users increases in the system.

The previous two chapters provide the offloading enabling technique and the system framework, with context-aware code offloading and scheduling algorithms for the proposed heterogeneous mobile cloud system to achieve near-optimal offloading performance. As an offloading service provided by wireless ad-hoc network of mobile devices, fault tolerance and reliability are among the most challenging issues. In the next chapter, we focus on developing a standalone fault tolerant mechanism for the proposed mobile cloud system mCloud to improve the its service reliability and provide a more complete failure tolerant solution to the fault tolerant module of mCloud.

Chapter 5

A Group-based Adaptive Fault Tolerant Mechanism

The availability and mobility management of mobile devices in the network can significantly hinder the performance of mobile cloud systems due to the frequent system faults caused by dynamic changes, and prevent applications from offloading to mobile ad-hoc networks. In order to improve the mobile cloud service reliability, in this chapter, we propose a group-based fault tolerant mechanism GFT-mCloud that classifies mobile devices into groups based on its processing capacity, mobility, and reliability. Different fault tolerance techniques are then devised adaptively based on the task offloading schedules and the specific group of machines it's offloaded. GFT-mCloud is designed as a standalone module that can work with existing mobile cloud code offloading systems. Extensive experiments have been conducted to evaluate the proposed mechanism. The results show that our fault tolerant mechanism is able to outperform conventional fault tolerant algorithms in the mobile cloud offloading environment.

5.1 Introduction

MOBILE cloud systems consist of mobile devices and servers connected via wireless networks, the mobility of the devices and intermittent network connections have become the major issues that induce system failures and further hinder the system reliability. Many fault tolerance solutions have been proposed to tackle the reliability issue in distributed computing systems, e.g. computer grids, mobile grids, and cloud computing systems. These approaches mostly consider a homogeneous computing envi-

This chapter is derived from:

- **Bowen Zhou** and Rajkumar Buyya, "A Group-Based Fault Tolerant Mechanism for Heterogeneous Mobile Clouds," *in Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2017)*, Melbourne, VIC, Australia, November 7 - 10, 2017.

ronment composed of identical machines and wired network connections. As a result, the fault tolerant policies proposed are often onefold solutions that lack adaptivity to the heterogeneous computing environment. On the contrary, mobile clouds are often composed of devices and wireless networks that have different specifications such as CPU speed, data throughput and signal strength. Therefore, existing fault tolerance approaches may not be able to maintain a similar system reliability level on mobile cloud systems as they are on grid computing systems.

In this chapter, we aim to propose a standalone fault tolerant mechanism that can work with existing frameworks and utilizes task offloading schedules devised from existing mobile code offloading algorithms to maintain the reliability of mobile cloud systems.

In order to make the fault tolerant mechanism adaptive to the heterogeneous mobile cloud, we propose a group-based fault tolerant mechanism GFT-mCloud for mobile cloud systems using reactive fault tolerant techniques. The machines such as mobile devices, cloudlets and public cloud servers are classified into different groups based on their hardware properties such as processing speed, reliability, and battery lifetime. The proposed GFT-mCloud mechanism can dynamically adjust different fault tolerant policies for different resource groups as well as different critical level of mobile tasks in a mobile application to improve the system reliability with lower processing overhead from the redundancy incurred. The key contributions of this chapter are as follows.

- We present a machine grouping algorithm for dynamically classifying machines in mobile cloud systems into multiple groups that have machines with similar capabilities. The groups can then be utilized by the proposed fault tolerant mechanism to obtain customized fault tolerant policies. The machine grouping algorithm is designed to be expandable for more grouping criteria.

- We propose a standalone fault tolerant mechanism GFT-mCloud for mobile cloud systems based on mCloud. It aims to work with mobile code offloading algorithms, and is able to devise different fault tolerant policies adaptively for different machine groups with the consideration of minimizing the overhead generated by the redundancy.

- The proposed fault tolerant mechanism is implemented as a library that can be added into the existing Android mobile cloud offloading frameworks as well as simulation environments.

- We implement a simulation environment of heterogeneous mobile clouds based on BRITE [158] and Weka, and test the proposed algorithm with various experiment scenarios as well as the efficiency of working with existing code offloadng algorithms in mobile cloud computing.

The rest of this chapter is organized as follows. Section 5.2 discusses the related works on the fault tolerant approaches in grid computing, mobile ad-hoc networks, and mobile clouds. Then we present the system modelling for the fault tolerant mechanism in Section 5.3. With the help of the models, we propose the group-based fault tolerant scheduling algorithm for HMC in Section 5.4. Section 5.5 presents the evaluation and discussions on the experimental results. Finally, a summary of this chapter is presented in Section **??**.

5.2 Related Work

Checkpointing and replication are two commonly used reactive fault tolerance techniques [221]. Checkpointing periodically takes snapshots of applications that contain running states and saves snapshots on reliable storage. The time between checkpoints depends on the system reliability. On the contrary, replication does not require state saving. Instead, it runs the replication of an application simultaneously on multiple computing resources to ensure task complete execution percentage meets certain level.

5.2.1 Fault Tolerance in Distributed Computing

Distributed computing paradigms include cluster computing, grid computing, mobile grids, etc. The loosely coupled computing resources connected via networks require fault tolerance management to maintain the system reliability. Dobber et al. [57] compared the performance of job replication (JB) and dynamic load balancing (DLB) on distributed system robust level. They provided a threshold value Y of Y^* based on job execution time that JB outperformed DLB when $Y < Y^*$. Naksinehaboon et al. [163] proposed an incre-

mental checkpoint and restart model for high performance computing (HPC). To reduce the overhead of checkpointing, the model aims to perform a set of incremental checkpoints between two full checkpoints by only saving address space that has changed since the last checkpoint. Noticeably, the fault tolerance in distributed computing systems only considers machine crash failures since the wired networks are stable. However, for a wireless computing system such as mobile cloud, device mobility and network link stability are of concern. Litke et al. [142] presented a replication-based algorithm, which utilizes the Weibull distribution for mobility analysis to estimate the number of replicas in order to maintain a certain level of fault tolerance for mobile grids. Most of the existing algorithms only apply the onefold fault tolerance policy, which is impractical for resource-scarce mobile devices involved computing systems as there is no guarantee for available computing nodes.

5.2.2 Fault Tolerance in Mobile Cloud Computing

A few fault tolerance algorithms were proposed for mobile cloud computing based on grouping. Chen et al. [33] proposed the k-out-of-n reliability control method to achieve energy efficiency and maintain system reliability level for data storage and processing in mobile cloud. The failure probability of a node is estimated by three factors: remaining battery, node mobility, and application factor. Park et al. [172] presented a fault tolerant algorithm for mobile cloud resource management. Different fault tolerance techniques such as checkpoint and replication are applied to different groups based on its availability and mobility. However, the grouping algorithm only considers device mobility and utilization rate. It is inextensible for more criteria. Choi et al. [42] classified devices in mobile grids into groups based on similar hardware properties. For tasks dispatched to low reliability groups, task replication is applied for fault tolerance, whereas the tasks dispatched to high reliability groups are migrated to another device upon failures.

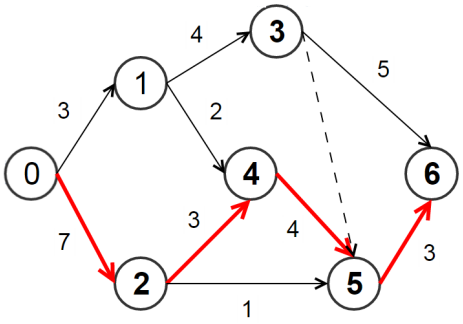
5.3 System Modelling

In this section, we formally present the models of the mobile cloud system, including application models, machine models, and the mobility model. We consider the mobile

cloud computing system as a network of heterogeneous machines consisting of mobile devices, nearby cloudlets, and public cloud VMs. The machines are loosely connected via various types of wireless networks such as Bluetooth and WiFi. The mobile devices are free to join and leave the network at any time while the cloudlets and public cloud VMs are considered stable. Therefore, the failures considered by this work are: 1) Machine crash failures, where machines either halt or do not response; 2) Loss of connection failures, where either wireless connections are disrupted or machines move out of the range of the mobile cloud network. The Byzantine failures [127] are not the focus of this work.

5.3.1 Application Model

Each mobile device in the network runs a mobile application where its tasks are evaluated to either offload to another machine or not. An application is modelled as a directed acyclic graph (DAG) $A = \langle S, E \rangle$ (example shown in Figure 5.1), where S represents the set of tasks s_i within the application, and E represents the dependencies of the tasks. The weight of each edge denotes the execution time of its pointed task. For instance, task s_1 in Figure 5.1 takes 3 time units to complete after task s_0 completed. It is assumed that a mobile task cannot be divided into subtasks and that it needs to be executed as a whole on a single processor of the machine. Each task is associated with a time variable $T_{tf}^{s_i}$ called total float, which is the amount of time a task can be delayed without delaying the completion of the entire application. $T_{tf}^{s_i}$ of task s_i can be obtained by calculating the difference between its latest finish time and earliest finish time. Then the critical path of a DAG can be identified as a path of tasks with 0 total float, which indicates the longest task execution time, i.e., the earliest finishing time of the application [102]. The rest of tasks can be finished as late as its total float and will not affect the completion time of the application. Therefore, tasks on critical paths of an application should be applied with fault tolerant policies to ensure the application to complete on time.



Task	Dependency
1	0
2	0
3	1
4	1, 2
5	2, 3, 4
6	3, 5

Figure 5.1: An example of application model. Critical path is marked in red

5.3.2 Machine Model

We consider the mobile cloud computing infrastructure M as a network of n heterogeneous machines. $m_i \in M(i = 1, 2, \cdots, n)$ denotes the ith machine. The term "machine" refers to either mobile devices, cloudlets, or cloud instances. We model the hardware specifications of a machine as follows.

$$m_i \triangleq < \mu_i, \theta_i, I_i^{wifi}, I_i^{cell}, I_i^{bt}, B_i^{wifi}, B_i^{cell}, B_i^{bt}, T_i^{avail}, T_i^r > \tag{5.1}$$

μ_i represents the processing speed of the machine. θ_i represents the utilization rate of the processor. $I_i^{wifi}, I_i^{cell}, I_i^{bt}$ are binary indicators denoting if the machine is equipped with that type wireless interface. For instance, if $I_i^{wifi} = 1$, machine m_i has access to WiFi, otherwise if $I_i^{wifi} = 0$. Variables $B_i^{wifi}, B_i^{cell}, B_i^{bt}$ represents the bandwidth of the wireless medium on machine m_i. The network speed of each wireless medium is the product of bandwidth and binary indicators I_i, plus current network latency. T_i^{avail} is the earliest available time of machine m_i, which represents the current workload of the machine. T_i^r denotes the time between failures of machine m_i. It is obtained by modelling the mobility of the machine.

5.3.3 Reliability Model

In order to take into account the machine reliability, the available time of a machine is considered. Note that the study of mobility model and trajectory is not the focus of this chapter.

To obtain the available time of a machine, the mean time between failures (MTBF) is adopted as T_i^r. MTBF describes the expected time of a machine between failures. In this

case, it represents the expected operating time of a machine before it is disconnected from the network. The Weibull distribution is often applied to effectively represent the machine availability in distributed computing environments [191]. The MTBF of a machine can be calculated from the Weibull distribution that abstracts the machine's behaviour. The probability density function of a 2-parameter Weibull distribution is given by:

$$f(t) = \frac{\beta}{\eta}(\frac{t}{\eta})^{\beta-1}e^{-(\frac{t}{\eta})^{\beta}}. \tag{5.2}$$

The history of the machine connection time to the mobile cloud network is used to estimate the shape parameter β and scale parameter η of the Weibull distribution. It consists of records of the time of that machine established a wireless connection as well as the duration of that connection.

The estimation of Weibull distribution parameters are obtained by applying the linear regression method to the cumulative distribution function (CDF) of Weibull distribution:

$$F(x) = 1 - e^{-(\frac{x}{\eta})^{\beta}}. \tag{5.3}$$

Then Equation 5.3 is converted into a linear equation form as

$$ln(ln(\frac{1}{1-F(t)})) = \beta ln(t) - \beta ln(\eta). \tag{5.4}$$

Then the parameters can be calculated using median ranks and least-squares fit on the data points generated from the history of the machine connection time in the network.

5.4 Group-based Fault Tolerant Mechanism GFT-mCloud

The mobile cloud computing environment usually comprises heterogeneous mobile devices, virtual machine instances on the cloud, and intermittent wireless network connections. To cope with the dynamics and heterogeneity, in this chapter, we propose a group-based fault tolerant mechanism GFT-mCloud that classifies machines into different groups based on its capability and adjusts different fault tolerance policies for tasks offloaded to different groups of machines.

5.4.1 Machine Grouping Algorithm

The GFT-mCloud is designed to operate on the offloading schedules of existing task offloading frameworks in mobile cloud computing. The position of the proposed standalone fault tolerant mechanism with the existing mobile cloud offloading frameworks

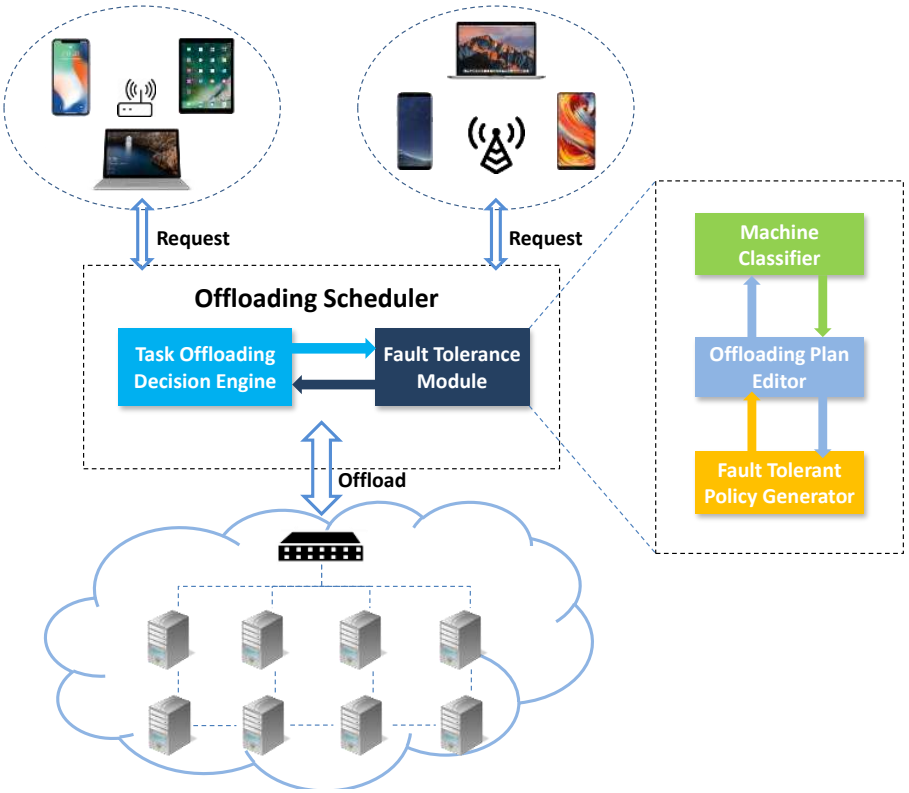


Figure 5.2: The proposed fault tolerant mechanism as a standalone module

is shown in Figure 5.2 and the proposed fault tolerant policies generating procedures in the view of one of the mobile devices are depicted in Figure 5.3. All the mobile devices follow the same procedures. The Code Offloading Scheduler is provided by the existing mobile cloud frameworks that have a decision engine to schedule the offloading tasks. We name the outputs of the decision engine as schedule plans. GFT-mCloud works as an add-on module to the framework. It includes a *Machine Classifier* for grouping machines, a *Policy Generator* for devising fault tolerant policy for each task, and a *Plan Editor* to decode the offloading schedule plan and encode the fault tolerant policy generated.

The grouping algorithm considers three criteria: machine processing capability, machine availability, and communication condition of the machine. In order to provide an automatic classification approach that can adopt an arbitrary number of criteria, clustering and decision tree learning are adopted in the machine grouping algorithm. Decision tree [60] is a tree-like graph that is comprised of interior nodes each representing a property variable, and leaf nodes each representing a class label. The path from the root node to each leaf node represents the values of the properties that classify the machines.

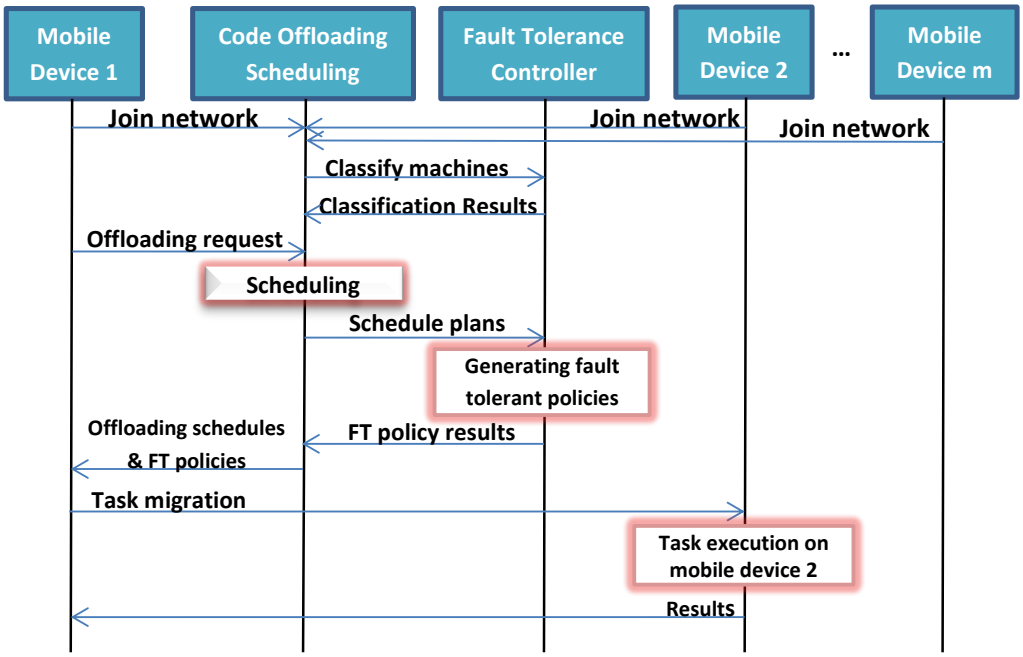


Figure 5.3: Interaction procedures of proposed fault tolerant mechanism

For each interior node, there is a set of split values that divide the set of objects into different groups. Moreover, the property variable on each level of the decision tree needs to be selected in order to provide an accurate classification. The decision tree structure can be trained from a set of supervised object data with the property values and its class label. The machine grouping algorithm includes two phases: data pre-processing phase and decision tree generating phase.

Data Pre-processing

First, the data set of machine instances with property values needs to be labelled so that it can be used to train the decision tree. The k-means clustering method [151] is used for labelling the data set with sub-classes based on each of the three criteria. The three criteria are further set with sub-classes based on the characteristics of the mobile cloud system. The machine processing capability criteria (**P**) is divided into slow, medium and high processing speed. The machine availability criteria (**A**) is divided into low and high since the machines in mobile cloud systems are mostly mobile devices and stationary machines. The communication condition criteria (**C**) is divided into poor and good group. The types of criteria and number of sub-classes of criteria can be set by users

in order to cooperate with the requirements of different executing conditions. In our case, a total of 12 labels are considered.

Second, each criteria needs to be quantified based on the property values from the machine data set. The processed data set is named as *learning set*. The processing capability is quantified by calculating the CPU speed of the machine and its utilization as follows.

$$P_i = \mu_i * \theta_i, \tag{5.5}$$

where P_i is the processing capability of machine m_i, μ_i is the CPU speed, and θ_i is the utilization of the CPU.

For machine availability, the algorithm considers the device mobility and the remaining battery lifetime of the device. Based on the mobility model described in Section 5.3, the device mobility is represented by its available time in the mobile cloud network. Since the longer the available time of device there is and the more battery there remains, the higher its availability is, the availability is defined as follows.

$$A_i = \alpha_{avail} T_i^r + \beta_{avail} \rho_{battery}, \tag{5.6}$$

where T_i^r is the device available time obtained from the mobility model, and $\rho_{battery}$ denotes the remaining battery percentage. $\alpha_{avail}, \beta_{avail}$ are weight factors that can be adjusted based on user preferences, and $\alpha_{avail} + \beta_{avail} = 1$. The two components of the equation are normalized to a range of 0 to 1.

The communication condition reflects on the data throughput of the wireless networks in mobile cloud systems. For both checkpointing and replication policy GFTmCloud considers, it requires fast communication and low energy consumption to constrain the incurred overhead. Therefore, the direct connections to other machines as well as the speed of the wireless connection are both considered. The communication condition is quantified as follows.

$$C_i = \sum_w (\alpha_{conn} B_i^w + \beta_{conn} * \frac{k^w}{n}) * I_i^w, \tag{5.7}$$

where B_i^w is bandwidth of wireless medium w, k^w is number of machines that machine m_i directly connected to via wireless medium w, and n is the total number of machines in its current mobile cloud network. The bandwidth of the wireless link B_i^w explicitly represents the data transmission speed. k^w reflects the level of redundancy machine m_i can achieve in the network as the more machines it has direct connections to, the more

Algorithm 4 Generate Decision Tree

1: **procedure** GETDECISIONTREE(S, C, V, L)
2: $n \leftarrow$ *create a root node for the tree*
3: **for all** $a_i \in A$ **do**
4: $en_i = entropy(S, a_i)$
5: $Gain(a_i) = entropy(S) - \sum_{v \in V} \frac{|S_v|}{|S|} en_i$
6: $a^* \leftarrow MAX(Gain(a_i))$
7: *assign attribute* a^* *to node* n
8: $S_{v_j} \leftarrow$ *seperate* S *based on values in* $V_{a^*}, v_j \in V_{a^*}$
9: **for all** S_{v_j} **do**
10: **if** S_{v_j}*is empty* **then**
11: *mark* S_{v_j} *as a leaf node and end the branch*
12: **else if** S_{v_j} *only contains items of label* $l_n \in L$ **then**
13: *put* l_n *as the leaf node and end the branch*
14: **else**
15: $A = A - a^*$
16: $ID3(S_{v_j}, A, V)$
17: *return* $T \leftarrow ID3(S, A, V)$

machines can be chosen for its fault tolerance redundancy. $\alpha_{conn}, \beta_{conn}$ are the weight factors that can be adjusted by the system, and $\alpha_{conn} + \beta_{conn} = 1$. The two components of the equation are normalized to a range of 0 to 1.

Machine Grouping Algorithm

The machine grouping algorithm operates on the *learning set* with ID3 (Iterative Dichotomiser 3) algorithm [182] to generate the decision tree for further machine grouping. The pesudocode is shown in Algorithm 4.

S represents the *learning set*, C is the set of criteria (in this case, three criteria considered), V denotes the set of split values for each criteria, and L is the set of class labels. Algorithm 4 returns the decision tree T for machine grouping. The algorithm first creates a root node for the tree. Then the criterion with the maximum entropy value is selected as the criteria for the root node (step 1-7). The entropy is used to evaluate differences between groups and is calculate as $entropy(S, a_i) = \sum_{v \in V_{a_i}} -v log_2(v)$. The items in S are separated into sub-groups V_j based on split values V_{a^*} of criteria a^* (step 8). For sub-groups V_j (step 9-18), if it is empty then marked as an end node; If it only contains items of label l_n, then make a node of this branch as label l_n; Otherwise, remove criteria a^* from

A and run $ID3$ with the sub-group S_{v_j} iteratively until the items in S are classified. The decision tree generated by Algorithm 4 can be updated with more machine data obtained or new criteria need to be added.

5.4.2 Group-based Fault Tolerant Algorithm

Due to the heterogeneity of mobile cloud system, a onefold fault tolerant policy approach may not adapt different execution conditions. For instance, applying replication method to the group with high reliability machines incur more operating overhead than checkpointing. On the contrary, applying checkpointing to low reliability machines can introduce unnecessary storing snapshots of checkpoints. Moreover, as the task offloading decision modules in mobile cloud systems have already devised an offloading schedule for mobile tasks, the fault tolerance algorithm needs to consider the execution location of the task being offloaded to maintain the offloading benefits as well as load balance.

In order to tackle this issue, we propose a group-based fault tolerance algorithm that takes into consideration the properties of different machine groups and adaptively select either checkpointing or replication as fault tolerant policy for the task based on the group of the machine the task is offloaded onto. The proposed fault tolerant mechanism operates on the task offloading schedules made by the mobile cloud frameworks in order to make the mechanism adaptive to the existing mobile cloud frameworks. Furthermore, the mobile tasks of different mobile applications on the devices are differentiated if they are on the critical path in order to adjust the fault tolerance policies.

Since the proposed fault tolerance mechanism adaptively applies checkpointing and replication methods, the number of replications as well as the frequency of checkpointing need to be determined.

Replication

The task replication is sent to machines in the same group of the original scheduled machine where the task is offloaded in order to maintain the offloading benefit in terms of execution time and energy consumption. Too many replications can incur large resource redundancy with high energy overhead. To overcome this issue, the machine for replicas is selected by ranking the machines in the group by reliability. The machines are ranked based on the failure rate, the execution time for the replication and number of directly

connected machines.

$$rank(m) = \alpha T^m_{comp} + \beta p^m_{fail} + \gamma \frac{N_{conn}}{N_{group}},\ \alpha + \beta + \gamma = 1, \tag{5.8}$$

where α, β, γ are weight factors can be adjusted by users, T^m_{comp} is the normalized task completion time on machine m in the scale of 0 to 1, N_{conn} denotes the number of machines in the group that machine m has direct connections with, N_{group} is the total number of machines in the current group, and p^m_{fail} is the task execution failure rate calculated as:

$$p^m_{fail} = \frac{N^m_{fail}}{N^m_{total}}, \tag{5.9}$$

and N^m_{fail}, N^m_{total} are the number of failed tasks and the number of total tasks executed on machine m. Then the machine with the lowest ranking score is considered the most reliable machine in the group and selected to deploy the replica of the task.

Checkpointing

The checkpointing method saves a snapshot of the running process periodically. The frequency of checkpoints is critical since it generates extra network traffic for the wireless communications and the checkpointing also incurs time overhead. Hence, the frequency needs to be determined based on the failure rate and the time taken by checkpointing. The method proposed in Young [247] is adopted to decide the frequency of checkpointing as follows.

$$T_c = \sqrt{2 T_s T_f}, \tag{5.10}$$

where T_c is the time interval between checkpoints, T_s is the time of checkpointing, and T_f is the time between failures.

The proposed fault tolerant algorithm is listed in Algorithm 5. TR represents the decision tree generated by Algorithm 4, M is the set of machines available in the mobile cloud network, and J is the set of tasks generated from mobile applications running on the mobile devices in M. First, TR is updated if more machine observations are captured in the system (step 2-3). Then, all the machines in M are classified into different groups through decision tree TR (step 4-5). Moreover, the tasks from each application are divided into critical and uncritical tasks using critical path analysis (step 6). After the machines and tasks are classified, the fault tolerance scheduler waits for an offloading decision of a task made by the offloading decision module of the system. Upon receiving the task with the offloading decision, the proper fault tolerance policy is selected (step

Algorithm 5 Group-based fault tolerance algorithm

```
1:  procedure GROUPBASEDFT(TR, M, J)
2:      if TR needs to update then
3:          TR ← ID3(S, A, V)
4:      for all m ∈ M do
5:          G_TR ← Classify(TR, M)
6:      {C, UC} ← classify the critical path of tasks in J
7:      while s ∈ J is scheduled by the offloading algorithm do
8:          g ← G_TR(s)
9:          if s ∈ UC then
10:             if T^s_comp < T^s_tf then
11:                 resend task s to originally scheduled machine
12:                 if machine is not available then
13:                     send s to the machine with EFT in g
14:             else
15:                 goto 17
16:         else if s ∈ C then
17:             if g contains L reliabiltiy property then
18:                 calculate ranks of machines in group g
19:                 m* ← lowest ranking machine
20:                 m* ← s_r
21:             else if g contains H reliabiltiy property then
22:                 T_c ← calculate checkpoint frequency
23:                 s ← insert checkpoint with T_c
```

7-23). First, the group label g of the machine that task s scheduled is identified (step 8), then the task is identified whether it is a task on the critical path of its application. If it is uncritical and the total float T^s_{tf} is long enough for re-execution (step 10), the task will be resent to the machine scheduled by the offloading decision module upon receiving a failure occurrence. When the machine is not available, the task will be scheduled to the machine within the same group that has the earlier finishing time (EFT) for the task (step 9-15). If the task is on its application's critical path, the fault tolerance policy applied is based on the group of the machine that the task is scheduled. If the group is with **L** reliability property, replication is applied. The replica of tasks is sent to the machine with the highest reliability ranking (step 17-20). If the group is with **H** reliability property, checkpointing is applied. The checkpoint frequency is calculated based on Equation 5.10 (step 21-23).

The time complexity analysis of GFT-mCloud.Assume there are m machines in the

Table 5.1: The configuration for simulations

Workload	**Application:** Optical Character Recognition, Chess game **Number of applications:** 50 **Computation length (No. of instructions):** Uniform(50000, 100000) **Data size (MB):** Uniform(0.5, 10)
Machine	**Processing Speed (MIPS):** Uniform(5000,200000) **Total available time (s):** Uniform(1000,30000) **Failure time point:** Weibull distribution($\alpha = 1.25, \beta = 92.74$) **WiFi bandwidth(MBps):** Uniform(1.0,1.2) **Bluetooth bandwidth(MBps):** Uniform(0.2,0.3) **Cellular bandwidth(MBps):** Uniform(0.8,0.9) **Number of machines:** Uniform(20,50)

network, and n tasks pass through the fault tolerance algorithm. The machine classification (step 4-5) takes $O(m)$ for grouping. For the fault tolerance policy selection part, the checkpoint frequency calculation (step 21-23) takes $O(1)$ and machine ranking calculation for replication costs $O(m)$. Therefore, the worst case scenario for selecting fault tolerance policy for n tasks costs $O(mn)$. Hence, the overall time complexity of the proposed algorithm is $O(m + mn)$.

5.5 Performance Evaluation

In order to evaluate the performance of the proposed fault tolerant mechanism, a series of simulations are conducted. The application completion time (i.e. makespan), the mechanism running overhead, and the number of control messages for fault tolerance are the three metrics for evaluation. Results from the proposed algorithm are then compared with other conventional fault tolerance algorithms.

5.5.1 Experimental Setup

We implement a simulator based on BRITE [158] to simulate the mobile cloud networks. BRITE is able to generate realistic Internet topologies with different bandwidth and delay values for each link of the topologies. We also adopted Weka [235] in the simulation to implement the machine learning methods in the proposed grouping algorithm. Table 5.1 lists the configuration for the simulation.

Due to the lack of real-world traces that are suitable for the heterogeneous mobile cloud environment described above, the workloads used in the simulation are obtained by profiling an OCR application and a chess game application running on mobile devices in our experimental environment. The workload contains a set of applications represented by randomly generated DAGs. Each vertex in the DAG represents a task of the application, and has two values, the amount of computation and data size. The two values are generated from uniform distribution with the range obtained from the application profiling.

The parameters used to characterize the machines in the experimental environment are also listed in Table 5.1. MIPS (Million Instructions per Second) is adopted in the simulation to represent the processing speed of the machine. It can be profiled by running a multi-thread integer computation looping program with a number of instructions and estimating the elapsed time upon completion. In order to simulate the machine failures, a 2-parameter Weibulll distribution is used for randomly generating the time between failures, and the failure time points are calculated by appending the time generated from the distribution to the machine start time. To capture the mobility patterns of real mobile devices in an area, the CRAWDAD trace sets [123] are used to obtain the shape and slop parameters of the Weibull distribution. The trace set is composed of system logs of WiFi access points on Dartmouth campus from September 1st, 2005 to October 4th, 2006. The number of sessions in every minute is extracted from the trace to generate the probability functions using rank regression.

5.5.2 Design and Implementation of Simulator

We implement a simulator based on BRITE [158] to simulate the mobile cloud networks. BRITE is able to generate realistic Internet topologies with different bandwidth and delay values for each link of the topologies. We also adopted Weka [235] in the simulator to implement the machine learning methods in the proposed grouping algorithm. In this section, we explain the design and implementation of the simulator and the library that can be used to implement the proposed mechanism on the Android platform.

The simulator developed consists of two parts: a set of simulator for mobile applications and mobile devices, and a library that implements the GFT-mCloud functions.

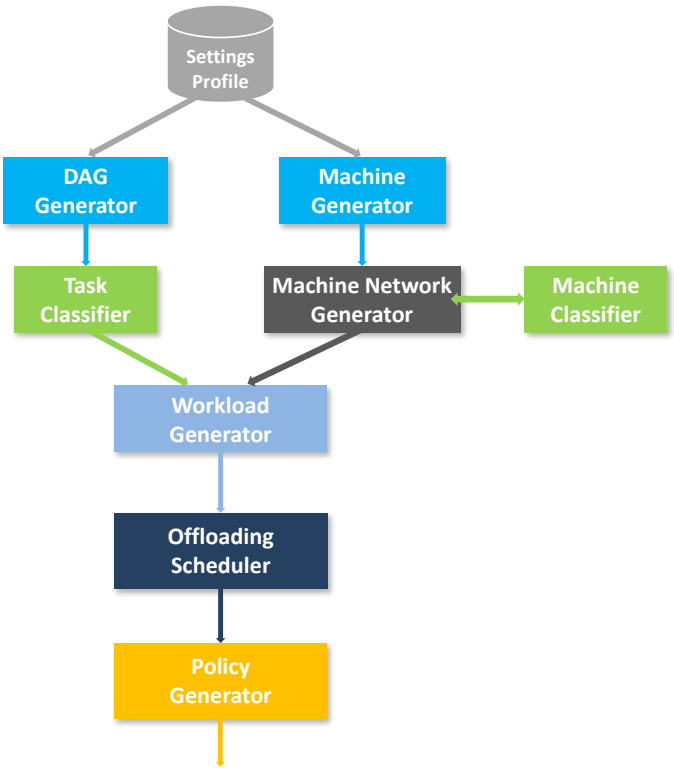


Figure 5.4: Implemented modules for the simulator

Figure 5.4 illustrates the design of the simulator. The set of simulator tools include DAG generator, machine generator, machine network generator, and the workload generator. The library implements the functions of task classifier, machine classifier, offloading scheduler and the fault tolerant policy generator based on the proposed algorithms in the GFT-mCloud.

The settings profile is used to store the simulation settings (shown in Table 5.1), and can be read by the multiple generators to produce simulation data. The DAG Generator generates random DAGs that simulate the mobile tasks and their dependencies within the mobile applications, and the Machine Generator outputs information of mobile devices from the device profile based on the simulation requirements. These two generators are implemented using Uncommons Maths library[2] for statistic distributions. The Machine Network Generator is for producing random mobile device networks in the HMC environment for simulation. The function is implemented using BRITE[3] to generate ran-

[2]https://maths.uncommons.org/
[3]http://www.cs.bu.edu/brite/

dom network topology with random connections between nodes and random link speed of the connections. Task Classifier and Machine Classifier are implemented using the Weka[4] library for decision tree methods.

At the start of the simulation, DAG Generator and Machine Generator read the settings from the simulation profile. Then the Machine Network Generator takes the device information and produces an HMC network. The Task Classifier and Machine Classifier apply the classification of the DAGs and devices in the machine network with group class labels based on the proposed machine grouping algorithm. Then the Workload Generator takes in the classified DAGs and machine network to randomly assigned tasks to machines as the tasks' generating location. At last, the Offloading Scheduler implements existing mobile cloud offloading algorithms, and the Policy Generator applies the fault tolerant policies to the offloading schedules based on the proposed mechanism.

5.5.3 Evaluation Results and Analysis

Machine Grouping Algorithm Performance Evaluation

In the first set of experiments, the accuracy of the proposed machine classification algorithm with multiple machine instances is evaluated. In order to improve the accuracy of the decision tree model proposed, the training set of machine instances should cover as many types of machine instances as possible. Since MIPS is used to represent the processing speed of machine (reflecting on CPU, cache and memory) that is either mobile device, or remote servers, we adopted the MIPS profiles obtained from CPU benchmarks [234], which include CPUs for desktops, laptops, and mobile devices. The overall available time of a machine is randomly generated from a uniform distribution. The network of machines in the mobile cloud system is generated by BRITE with different bandwidth on different links to represent multiple types of wireless medium. For the validation sets, three sets of machine instances are extracted from the real-world traces. The first set is a synthetic set that is generated from the same data distribution as the training set. The second and third validation set are extracted from the CRAWDAD trace.

Three features of the machines are considered in the machine grouping algorithm

[4]https://www.cs.waikato.ac.nz/ml/weka/downloading.html

Table 5.2: Results of the cross validation

Item	Synthetic Set	CRAWDAD Set 1	CRAWDAD Set 2	CRAWDAD Set 3
Correctly Classified Instances	997	1397	3986	3575
Incorrectly Classified Instances	5	3	14	7
Mean absolute error	0.0008	0.0004	0.0009	0.0003
Relative absolute error	0.6503%	0.2777%	0.6847%	0.2133%
Total number of instances	1002	1400	4000	3582
Accuracy	99.501%	99.786%	99.649%	99.805%

when classifying machines: processing speed, availability, and communication capacity. Processing speed is split into three sub-groups: fast, medium, and slow. The availability and communication capacity are both split into two sub-groups: high and low. The weight factors $\alpha_{avail}, \beta_{avail}, \alpha_{conn}, \beta_{conn}$ in Equation 5.6 and 5.7 are all set to 0.5. Note that these weight factors will not influence the performance of the classification since both training sets and validation sets are using the same weight factor configuration to quantify machine features. The results of cross validation using WEKA are reported in Table 5.2. As we can observe from the WEKA results, the grouping algorithm gives an accuracy of 99.5% on the synthetic machine instances and for the three real-world trace machine instances, it gives 99.79%, 99.65% and 99.8% accuracy respectively. Also, the reported accuracy remains similar while the number of instances grows. These results show that the proposed machine grouping algorithm can accurately classify the machines into different groups based on the three features.

Group-based Fault Tolerant Mechanism Performance Evaluation

First, the weight factors α, β, γ used in the models (Equation 5.8) are set in different combinations to test their effects on the fault tolerance performance. The configurations of weight factors are listed in Table 5.3. The same network topology that contains a total number of 30 machines are used as the simulation environment. 50 randomly generated application DAGs are generated as the testing workload. The task offloading schedules of the application DAGs among the machine network used are obtained from the list scheduling heuristic HEFT (Heterogeneous Earliest Finish Time) [220] for the remaining

Table 5.3: Weight factor setting cases

Case	α	β	γ	Case	α	β	γ
1	1	0	0	7	0.2	0.6	0.2
2	0.8	0.1	0.1	8	0.2	0.2	0.6
3	0.6	0.3	0.1	9	0.1	0.7	0.2
4	0.6	0.1	0.3	10	0.1	0.2	0.7
5	0.4	0.4	0.2	11	0	1	0
6	0.4	0.2	0.4	12	0	0	1

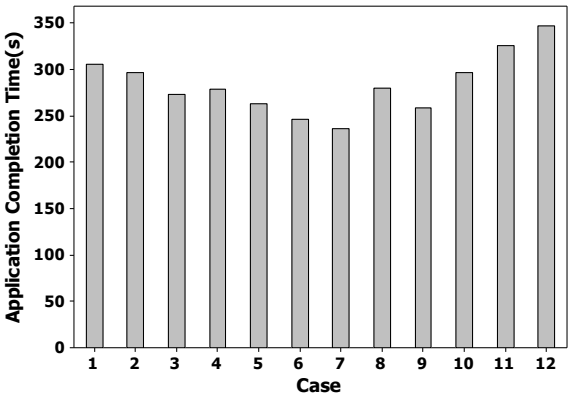
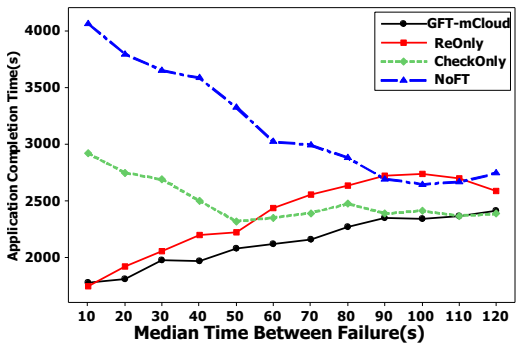


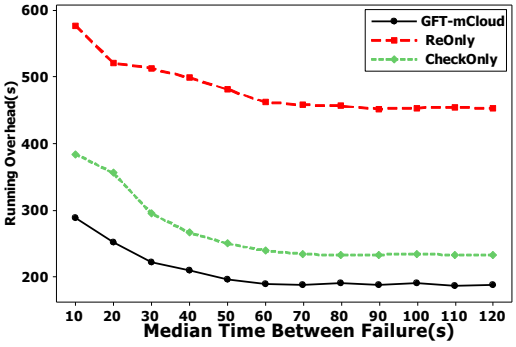
Figure 5.5: Average application completion time for different weight factor cases

sets of experiments in this section. The schedule plans are then put through the proposed FT mechanism and the other three policies for experiments. The results of application completion time (ACT) are depicted in Figure 5.5. As the results show, the best weight factor setting with the lowest ACT is from case 7, which is $\alpha = 0.2, \beta = 0.6, \gamma = 0.2$. The average ACTs slightly decrease as the value of α decreases and the value of β increases. However, the ACT begins to increase again when the value of γ increases from 0.5 (case 6,8,10,12). These results indicate that the weight factor β that represents the weight of a machine's number of failures dominates the efficiency of the replication, and further influences the performance of GFT-mCloud in terms of overall application completion time. This is because when the mechanism selects replicas based on equation 5.8, the more reliable the machine is, the less probability that the replication fails to complete, which leads to a lower application completion time. The weight factor setting $\alpha = 0.2, \beta = 0.6, \gamma = 0.2$ will be used for the rest of the experiments in this section.

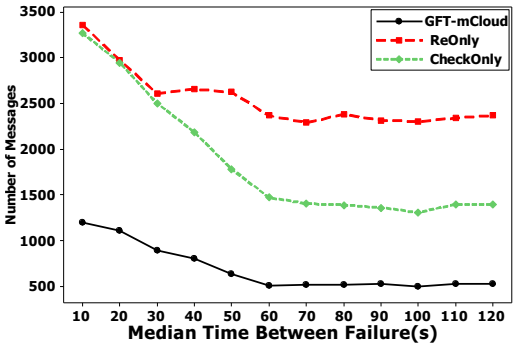
Secondly, we evaluate the GFT-mCloud in terms of the adaptivity on generating different fault tolerant policies based on different types of machines and tasks. The proposed mechanism is tested with 500 randomly generated application DAGs and machine



(a) Overall application completion time



(b) Running overhead



(c) Number of messages for fault tolerance management

Figure 5.6: Performance under different machine availability

networks, and results are averaged with 50 runs. Each DAG is generated with a random number of vertex as tasks and edges as task dependency. The tasks are coupled with data size and the amount of computation from the application profiling. The results are then compared with three other fault tolerance baselines: 1) replication only policy (**ReOnly**), 2) checkpointing only policy (**CheckOnly**), and 3) no fault tolerant policy (**NoFT**). **ReOnly** only applies replication of tasks as fault tolerance solution, while **CheckOnly** only uses checkpoint approach to maintain the task completion reliability. Three test cases are conducted below to evaluation the fault tolerance performance of GFT-mCloud. **Case A** study the influence of machine availability reflecting on its median time between failures. **Case B** and **case C** consider the effect of task computation and data size respectively on fault tolerance performance of the GFT-mCloud, since these two factors may affect the mobile cloud offloading.

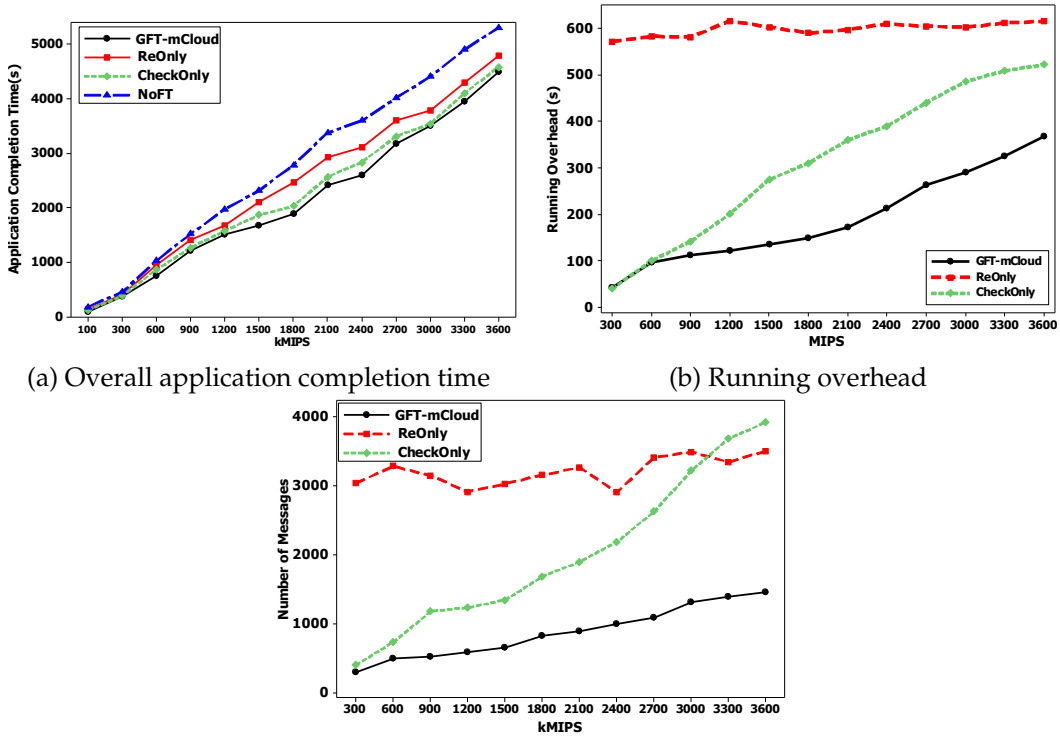
Test case A. In the first case, the performance of the proposed FT mechanism for different machine availability is evaluated. The machine availability is represented by the

median of time between failures (MTBF) of the machines. The failure time points of each machine are generated from the Weibull distribution above-mentioned. The average application completion time of different machine availability are shown in Figure 5.6a. As we can observe, GFT-mCloud outperformed the other three baselines. When the median time between failures is as small as 10 and 20 (i.e., machines are highly unavailable), the NoFT baseline generates the worst completion time of more 4000 seconds. CheckOnly generates the second worst completion time of around 3000. On the contrary, GFT-mCloud and ReOnly generated the application completion time below 2000, which is the lowest among the four algorithms compared. This is due to the fact that when the failure happens more frequent, the NoFT baseline and CheckOnly baseline keeps restarting the task execution, which leads to a higher overall application completion time than proposed mechanism and ReOnly baseline.

Moreover, when the machines have frequent failures, GFT-mCloud tends to apply replication as the policy, which makes its results similar to the ReOnly baseline when the MTBF is low. As the MTBF grows, the application completion time of NoFT and CheckOnly baseline decreases, and GFT-mCLoud and ReOnly baseline increases. This is because as the machine can execute tasks for longer time, the number of restarts for NoFT and CheckOnly becomes less as well as the remaining execution of the task if failures happen, which leads to a shorter makespan. Also, as machines become more available, the replicas are delayed to run on the backup machine since it needs to finish its own tasks first, which leads to a longer makespan for ReOnly. All four algorithms generate stable results when the median time between failures are greater than 90. This is because as the machines become more reliable, ReOnly baseline generates more redundancy overhead for replica transmissions than CheckOnly baseline (Figure 5.6b). As the median time becomes longer, most of the tasks complete before failures happen, which makes the completion time generated more stable.

As shown in Figure 5.6b and Figure 5.6c, the running overhead and number of messages incurred with the overhead are decreasing and become stable, which is consistent with the overall application completion time. The running overhead of the proposed FT algorithm accounts for around 9% after being stable.

Test case B. In the second case, the algorithms are evaluated with DAGs that have



(a) Overall application completion time

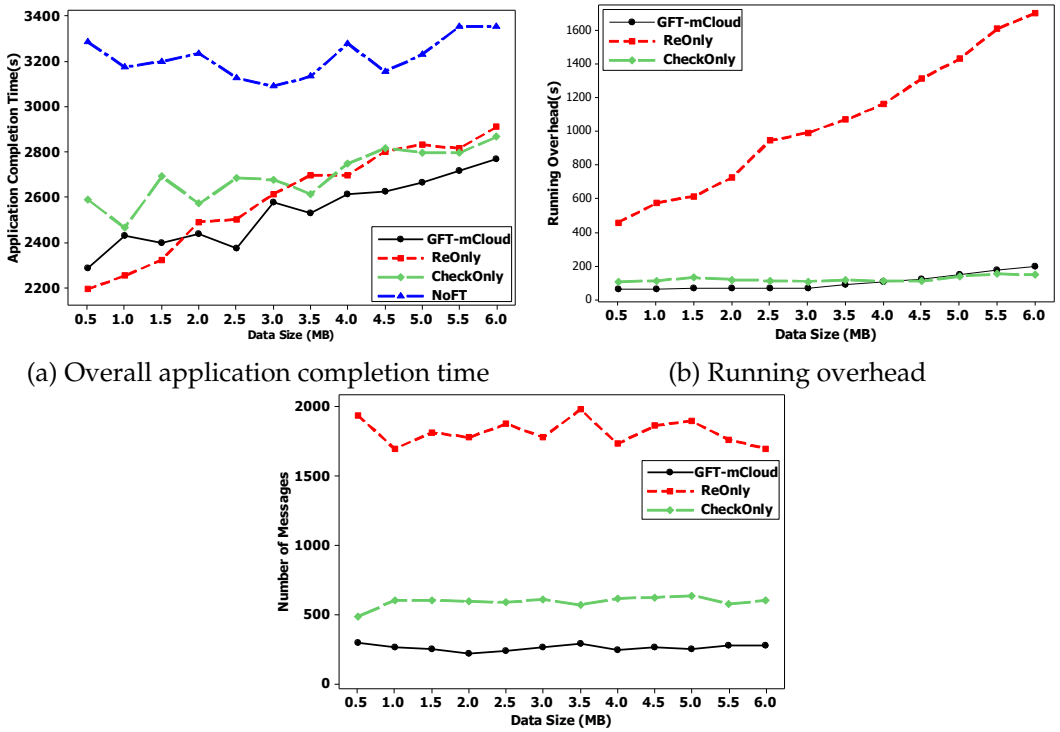
(b) Running overhead

(c) Number of messages for fault tolerance management

Figure 5.7: Performance under different task computation requirements

different amount of task computation requirements in terms of MIPS, to show the performance of the proposed FT mechanism with different applications. Machines used in this experiment are generated with the median time between failures from 90s to 120s. The average application completion time generated by four algorithms is depicted in Figure 5.7. When the computation amount is low, the application completion time generated by the four algorithms are almost same. As the computation requirement grows, the completion time grows in the similar speed, with NoFT giving the worst results among the four algorithms and GFT-mCloud outperforming the other three baselines. The growths of the completion time from the four algorithms are similar. This is consistent with the results shown in Figure 5.6 when the machine's median time between failures is around 90s to 120s. Additionally, GFT-mCloud performs the best when the average task computation requirement is above 1500 kMIPS as we can observe in Figure 5.7.

Additionally, Figure 5.7b and Figure 5.7c show the running overhead and control messages incurred by the redundancy of the fault tolerance management for GFT-mCloud,



(a) Overall application completion time

(b) Running overhead

(c) Number of messages for fault tolerance management

Figure 5.8: Performance under different task data sizes

ReOnly and CheckOnly baselines. For ReOnly baseline, the overhead and number of control messages have not fluctuated much under machine availability change, because ReOnly baseline generates replicas for all tasks in the application. However, the running overhead and control messages increased for GFT-mCloud and CheckOnly since more checkpointing snapshots were generated when the task execution is longer.

Test case C. In the third case, the performance of the proposed FT mechanism is tested with different amount of data (i.e., data size) to see its effect in case of task migrations due to failures. The median of task computation in the workload is 1800 kMIPS. Machines used in this experiment are generated with the median time between failures from 90 seconds to 120 seconds. Results from the proposed algorithm along with the three baselines are shown in Figure 5.8.

In Figure 5.8a, the makespans generated by NoFT, which are around 3300, do not fluctuate too much when data size grows. For GFT-mCloud and CheckOnly baseline, the makespan increased slowly as the data size grows, while the makespan of ReOnly

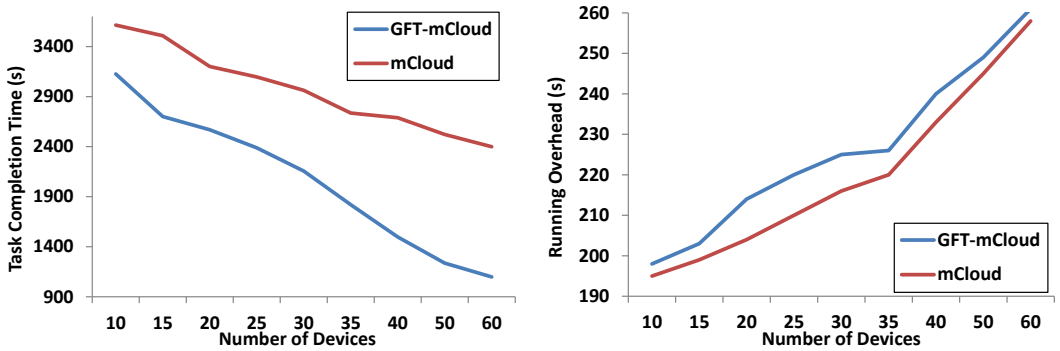
increased more rapidly. This is due to the growth of data size having a greater impact on replication as each task has to maintain its replica in the beginning, but only small impact on checkpointing as it only requires task migration when failure happens. It can be further observed from Figure 5.8b and Figure 5.8c that the data size growth has little impact on GFT-mCloud and ReOnly as the running overhead remain stable, while the running overhead of ReOnly increased due to the replication requires more time for migration and processing. Moreover, the number of fault tolerant related control messages for the three algorithms do not change much with the data size growth. Comparing to the results in Figure 5.6c and Figure 5.7c, the number of control messages will only be affected by the machine availability and task computation amount instead of data size.

Discussion. The results from four sets of experiments conducted showed that the proposed GFT-mCloud is able to adapt its policies to different execution conditions, such as different machine availability and different applications (computation requirement and data size), and can generate the lowest makespan among the algorithms compared with around 5-10% running overhead and a small number of control messages. The application computation intensity (Figure 5.7) has the most effect on the performance of the proposed FT mechanism, while the data size of applications have limited impact of around 8% (Figure 5.8) in terms of makespan.

Scalability of GFT-mCloud Mechanism

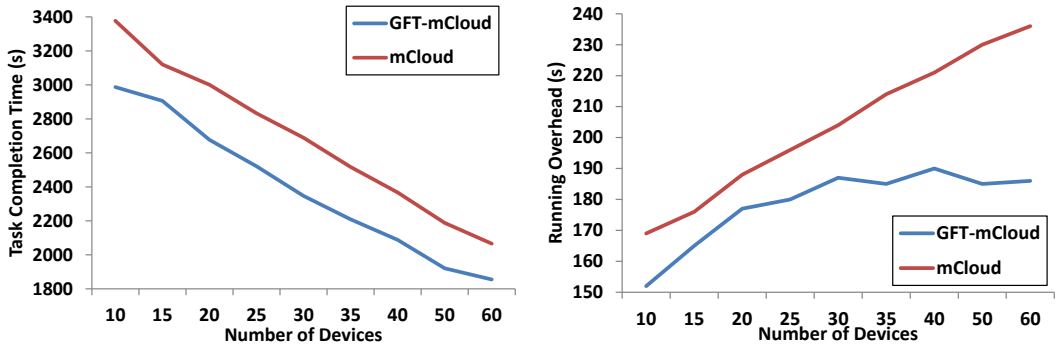
For this set of experiments, we evaluate the scalability of the proposed GFT-mCloud mechanism when the number of mobile devices and tasks increases. Since the heterogeneous mobile cloud environment is an ad-hoc network of mobile devices and other computing resources, it is possible that there would be a very fluctuant number of devices connecting to the GFT-mCloud enabled mobile cloud systems simultaneously. In addition, when there are more devices participating in the HMC network, the proposed mechanism should be able to utilize the added resources to provide equal or better fault tolerant performance. Therefore, the scalability of the mechanism in terms of fault tolerance are evaluated.

The evaluation metrics tested in this experiment is the average task completion time and the time overhead generated by the fault tolerant processing. 500 randomly gen-



(a) Overall application completion time (b) Running overhead

Figure 5.9: Scalability of GFT-mCloud with low machine availability



(a) Overall application completion time (b) Running overhead

Figure 5.10: Scalability of GFT-mCloud with high machine availability

erated DAGs with the amount of computation and data size drawn from the uniform distribution listed in Table 5.1 are used for running on the machine networks with different number of nodes. The machine networks are generated using the implemented BRITE machine network generator. The results are averaged by 50 runs. We compare the performance with low machine availability case and high machine availability case. The median time between failures for machine availability is drawn from the uniform distribution $(20, 50)$ and $(100, 120)$ respectively. The performance of the proposed algorithms in terms of task completion time and overhead are compared with the offloading algorithm in mCloud without fault tolerant policies applied, and are shown in Figure 5.9 and Figure 5.10.

Figure 5.9 shows the results of low machine availability scenario. As can be observed in Figure 5.9a, both results from GFT-mCloud and mCloud have shown declining task completion time due to the more mobile devices participating in offloading services.

However, comparing to mCloud, the results from GFT-mCloud have a faster decline when the number of mobile devices is increasing. This is because the mCloud offloading policy only restarts a mobile task when there is a fault happening, while the GFT-mCloud adaptively applies the checkpointing and replication to the mobile tasks. Regarding the process overhead, Figure 5.9b shows that the GFT-mCloud produces slightly more time overhead than mCloud, but it accounts for only a small fraction in terms of the overall task completion time reduced.

Figure 5.10 shows the results of high machine availability scenario. Comparing to results in Figure 5.9a, the task completion time with high machine availability has a slower decline as the number of devices increases, due to devices in the network have failures less frequently. It is also noticeable that the processing overhead shown in Figure 5.10b indicates that when the device availability is high, the proposed GFT-mCloud generates lower overhead that the mCloud, and as the number of devices increases, the overhead stays around 185 seconds. This is because checkpointing policy are mostly applied to mobile offloading tasks GFT-mCloud mechanism when the device is more stable, while the mCloud policy only restarts the task by sending another copy when failures happen.

5.6 Summary

The mobility of mobile devices, as well as the intermittent wireless connections, makes mobile cloud computing systems vulnerable to failures. As the existing fault tolerant algorithms for distributed systems only focus on machine crash failures, they do not suit mobile cloud environment that involves wireless network failures. In this chapter, we proposed a group-based fault tolerance mechanism that works as a standalone module with the existing mobile cloud offloading frameworks. It classifies the machines in the mobile cloud network into different groups based on its capabilities using decision tree method. Different fault tolerant policies such as checkpointing and replication are adaptively applied based on the offloading tasks and the group of its offloaded machine in order to select a machine within that group for redundancy. The experimental results have shown that the proposed fault tolerance mechanism is able to adapt different machine capability and generate stable makespans, with low running overhead.

Having provided solutions for mobile code offloading, load-balanced task scheduling and fault tolerant mechanism for the heterogeneous mobile cloud environment, there are still concerns that mobile users may lack incentive to use the proposed offloading services due to the battery lifetime concern. Therefore, in the next chapter, we design a reverse auction-based incentive mechanism that encourages mobile users to participate with rewards in the HMC offloading service.

Chapter 6

A Reverse Auction Based Incentive Mechanism

The previous chapters have introduced solutions to enable a load balanced, fault tolerant task offloading service in the heterogeneous mobile cloud environment. However, mobile users may be discouraged from sharing their devices for running foreign tasks due to the concern of battery lifetime and security. To incentivize mobile device users to utilize and participate in the proposed task offloading service, in this chapter, we design a task offloading market for the heterogeneous mobile cloud service, where a mobile user can compete as a seller with others by bidding its redundant computing resources, and another mobile user as a buyer can pay the bidding price and offload the task to the winning user. To enable an incentive and fair auction of the mobile cloud offloading market, we propose a reverse auction-based incentive mechanism, mCloudAuc, to provide real-time auctions. The proposed auction algorithm demonstrates computation efficiency, truthfulness, and individual rationality for the participants through proof and multiple simulations. Our prototype implementation mCloudAuc based on the proposed mCloud framework on Android platform has also shown its feasibility in practice.

MOBILE cloud computing is a computing paradigm that enables mobile devices to outsource their computation intensive tasks onto cloud computing resources for execution to conserve the battery on mobile devices and improve the performance of mobile cloud applications. Take optical character recognition (OCR) application for example, a mobile user can take a photo of a piece of paper containing text in a foreign language, and the app will process the photo and show the translated text on the display. Alternatively, the user can subscribe to the cloud service provided by the application provider by paying for the subscription fee to offload the photo to cloud services for processing, and release the device from halting. However, network bottleneck and lack of performance improvement of offloading to cloud have been the major problem of adapt-

This chapter is derived from: **Bowen Zhou**, Satish Srirama, and Rajkumar Buyya, "An Auction-Based Incentive Mechanism for Heterogeneous Mobile Clouds," *Journal of Parallel and Distributed Computing*, 2018 (Under review).

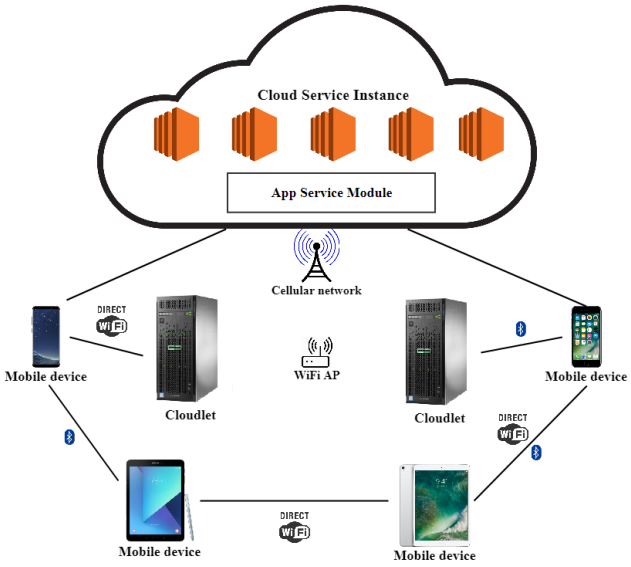


Figure 6.1: An example of a local mobile device network using heterogeneous mobile cloud services

ing the mobile cloud paradigm. For example, a chess game on the Samsung Galaxy S3 will only benefit from offloading to an m3.medium or more powerful instance on Amazon EC2, which generates more monetary cost and reduce the user incentive of using mobile cloud offloading services [210].

Figure 6.1 shows an example of the mobile cloud collaboration scenario in the heterogeneous mobile cloud environment (HMC), where the mobile devices in a local area form a wireless ad-hoc network through short-range wireless network such as WiFi, Bluetooth and WiFi-direct. In addition, some users also own cloudlets that are publicly connectible for other mobile devices via WiFi (e.g., a café owner), and some users have subscribed to the application's cloud service with dedicated instances running in the cloud. However, as an opportunistic network of computing resources, the mobile users may not be willing to provide their own devices to other users for running tasks due to the limited battery or privacy concerns. Therefore, the key to attaining full benefits of offloading in HMC is incentivizing all the mobile users to commit their computing resources to the mobile cloud offloading service.

In this chapter, we propose an incentive mechanism for heterogeneous mobile cloud offloading to encourage mobile users to participate and get appropriate rewards in return. Particularly, we design an offloading market where the mobile users who wish to

offload their mobile tasks to remote resources are considered as buyers who submit offloading requests to the market and the mobile users who have redundant resources are considered as sellers that can announce the prices they are charging and compete on the market to lease their resources to the buyers for offloading. Within the HMC environment, the challenges for the designed offloading market are:

- The mobile devices can be both buyer and seller, and each device may have different amount of computing resources since the devices are heterogeneous and can own other types of resources such as cloudlets and cloud instance subscriptions. Therefore, the execution cost for each mobile device to complete the same offloading task may be different.

- Since the objective of mobile cloud offloading is to conserve energy and execution time, the offloading decisions made by the proposed incentive mechanism need to guarantee these offloading benefits.

- As individual mobile users, sellers on the offloading market may try to gain more income by overpricing their offloading services, or lose income as a result of unfair competition and underpricing. Therefore, the proposed incentive mechanism needs to guarantee a fair market with truthfulness and individual rationality.

- Since mobile devices may leave the market before the offloading tasks they are serving are completed, the incentive mechanism needs to introduce penalty and fault recovery mechanism to reduce service disruptions.

In order to tackle these challenges, we propose a reverse auction based incentive mechanism for the designed offloading market in the HMC. The proposed mechanism consists of two parts: an auction algorithm that is responsible for allocating the offloading tasks (buyers) to the resource sellers and deciding the payments for sellers, and the penalty scheme for the failed sellers. The proposed auction algorithm aims to guarantee the trustfulness and individual rationality of the auction participants. The proposed incentive mechanism is evaluated under substantial simulations with real-world workload traces and real experiments with the prototype system implemented on the Android platform. In summary, the main contributions of this work are as follows.

1. We model the task offloading service in the heterogeneous mobile clouds as an offloading market where the mobile devices requesting to offload are considered as buyers and the mobile devices that offer their redundant resources for offloading are considered as sellers. Sellers compete on the market by bidding the different types of resources such as their mobile device resources and cloudlets or cloud subscriptions to other buyers.

2. We propose a reverse auction based incentive mechanism for allocating the buyers' requests to sellers' bids and a pricing policy to decide the payment from the buyers to the sellers. Also, a fault recovery and penalty mechanism as part of the incentive mechanism is proposed to cope with sellers who cannot complete the requests assigned.

3. The reverse auction based algorithm is designed as the online algorithm and has a low time complexity. It can guarantee the individual rationality and trustfulness of participants in the auction.

4. We implement a prototype of the incentive mechanism on top of the code offloading framework mCloud we proposed. The incentive mechanism is then evaluated with both simulations and real experiments on the prototype.

The rest of this chapter is organised as follows. We first discuss the related works for incentive mechanism in mobile cloud computing in Section 6.1. Then the system models and the problem formulation are given in Section 6.2. The design of proposed incentive mechanism and the economic property proofs are presented in Section 6.3, followed by a description of the prototype implementation and performance evaluation in Section 6.4. Finally, we give a conclusion in Section 6.5.

6.1 Related Work

6.1.1 Incentive Mechanisms in Mobile Opportunistic Computing

Incentive mechanisms are essential for a computing environment that involves opportunistic and volunteering mobile device users, such as Internet-of-Things (IoT) and mo-

bile crowdsourcing [23]. Tian et al. [217] designed the incentive mechanism based on the devices' movements, which motivates participants to move around and gather more sensing data. A greedy algorithm was proposed to solve the task allocation problem. Zhang et al. [251] proposed a multi-market dynamic double auction mechanism, MobiAuc, for the proximity-based mobile crowd service. MobiAuc addressed the overlapping multi-group mobile device problem, and it is trustful and individually rational. Wang et al. [230] presented a reverse auction formulation for quality-aware mobile crowdsensing system to minimize the overall expenditure of the system. A quality score is assigned to each user based on the availability, accuracy of sensor data, reputation, etc. A game-based incentive mechanism for multi-resource sharing was proposed by Gan et al. [74] to share their idle resources in mobile social crowdsourcing. A task allocation process, profit transfer process and reputation management process are combined to achieve a trustfulness and individual rationality for the proposed incentive mechanism. Liu et al. [147] studied the incentive mechanism for computation offloading in IoT by proposing a Stackelberg game-based approach. The approach was shown to be guaranteed to reach a unique Nash equilibrium. As we can observe from the related works, two main approaches for incentive in the voluntary computing network are auction based and game theory based.

6.1.2 Incentive Mechanisms in Mobile Cloud Computing

Only few works have studied the incentive issue for heterogeneous mobile clouds. One closely related work is presented by Jin et al. for HMC [105]. They considered the mobile devices as buyers only and the cloudlets and public clouds as sellers only. A two-stage double auction is designed to determine the winners and prices in buyers and sellers to allocation the offloaded tasks, and further select seller candidates for buyers who won more than one auction. Similarly, Xie et al. [239] designed a distributed multi-dimensional pricing mechanism based on game theory for cloudlet-based offloading allocation and scheduling. Three types of prices such as multi-dimensional price, penalty price, and discount price are designed to motivate resource sharing. Tang et al. [215] proposed a double auction based incentive mechanism for mobile ad-hoc network cloud with a single market-clearing price policy. Different user behaviour of price taking and

price anticipating are analysed with a game-theoretic approach.

The existing works have not studied the incentive issue in the proposed HMC environment where all devices can be both sellers and buyers. Moreover, the computation efficiency of incentive mechanisms is vital for mobile devices and auction mechanisms such as double auction have high time complexity and are hard to implement [162]. Hence, we proposed a reverse auction based incentive mechanism for HMC environment that satisfies trustfulness and individual rationality, with low computation complexity.

6.2 System Model and Problem Formulation

In this section, we first present the system model that abstracts the various characteristics of the proposed mobile task offloading market. Then, based on the system models, we formulate the task offloading problem in HMC as the offloading market problem in the form of integer linear programming. The objective of the problem is to maximize the income of the mobile device users for sharing resources, with the consideration of offloading benefits and the unique constraints in the HMC environment.

6.2.1 System Model

As depicted in Figure 6.1, there are three types of computing resources considered in our heterogeneous mobile cloud environment: mobile devices such as smartphones and tablets, nearby cloudlets like laptops, and public cloud instances managed by the mobile app provider.

Consider there are a number of m mobile devices on the market. The mobile devices have at least one of the wireless network interfaces available, including WiFi, cellular network, or short range mediums such as Bluetooth and WiFi-direct. All devices within the network are running the same mobile cloud application. The mobile device is modelled as follows.

$$m_i =< \mu_i, \theta_i, I_{wifi}, I_{cell}, I_{bt}, P_i^{active}, T_i^{avail} > \tag{6.1}$$

where μ_i represents the processing speed of device m_i, θ_i represents the utilization of the processor, $I_{wifi}, I_{cell}, I_{bt}$ are the binary indicator for the wireless medium availability, P_i^{active} is the energy consumption rate when the device is active, and T_i^{avail} is the device's

Table 6.1: Symbols of the system model

Notation	Description
m_i	mobile device
μ_i	processing speed of device m_i
θ_i	processor utilization of device m_i
$I_{cell}, I_{wifi}, I_{bt}$	binary indicator of the wireless medium availability of device m_i
P_i^{active}	active energy consumption rate of mobile device i
T_i^{avail}	available time of device i in the network
c_i	required amount of computation of an offloading task
d_i	deadline of an offloading task
S_j^{com}	cloud instance subscription
ω_j	processing speed of cloud instance subscription S_j^{com}
t_j^s, t_j^f	starting and finishing time of cloud instance subscription S_j^{com}
p_j^{com}	the price of the subscription S_j^{com}
$T_{ij}^{comp}, E_{ij}^{comp}$	execution time and energy consumption of task i running on device j
x_{ij}	binary decision variable for task i offloading to device j
y_j	binary decision variable for winner device j of the market

available time in the network.

The cloudlets are mobility-enhanced small-scale cloud data centres that are located at the edge of the Internet. Each cloudlet is considered as an always available and stable computing resource owned by one of the users on the market. The cloudlet is accessible via wireless networks. The hardware properties can be abstracted using the same model as the mobile device. The public cloud services are provided and managed by the mobile cloud application provider via app service module. It is enabled within the app in the form of subscriptions. The subscriptions contain service description including capacity, service time, and price. The public cloud instances will be dedicated to the user for the purchased period. The subscriptions can be modelled as follows.

$$S_j^{com} =< \omega_j, T_j^{com}, p_j^{com} >, \tag{6.2}$$

where ω_j represents the processing speed of the cloud instance subscription S_j^{com}, T_j^{com} is the purchased duration of purchased subscription S_j^{com}, and p_j^{com} denotes the price rate of the subscription.

We consider the task offloading in HMC as an offloading market, where the mobile user can run the application entirely on his local mobile device or offload the computation intensive tasks to other computing resources in the network which are sharing the

redundant resources. On the offloading market, the mobile users who wish to offload their computation intensive task are considered as buyers, while mobile users who wish to share their redundant resources are considered as sellers who compete with different prices to fulfil the buyers' offloading requests. The redundant resources can be the local resource of other mobile devices, nearby cloudlet resources owned by other users, and public cloud resources owned by the app users who have purchased them. In order to abstract the task offloading processes, the offloading request is modelled as follows.

The user who wishes to offload his tasks to other resources can submit a task offloading request to the auctioneer in the HMC network. The request contains the requested resources and the demand of the resource usage including requested amount of computation and task deadline.

$$r_i =< c_i, d_i >, \tag{6.3}$$

where c_i represents the demanded amount of computation for the offloading task, and d_i represents the deadline of the offloading task.

Similarly, the user who wishes to sell their redundant resources on the market can submit a service offer to the auctioneer in the network. The service offer contains the specifications of the resources, and it is modelled as follows.

$$s_j =< \omega_j, t_j^s, t_j^f, b_j > \tag{6.4}$$

where ω_j represents the processing speed of the service offers$_j$, t_j^s and t_j^f are the starting and finishing time of s_j, and b_j denotes the bid of the seller. b_j is determined by the seller, and it is supposed to be reported truthfully to his real valuation of the job with the help of the auction algorithm design.

At last, the cost of task execution is modelled in terms of exectuion time and energy consumption for the offloading tasks. Assuming there are a number of i task offloading requests and a number of j devices offering the offloading services, then the execution time for task i on the leased resource of device j is

$$\begin{aligned} T_{ij}^{comp} &= \frac{c_i}{\omega_j} + \frac{I_i + O_i}{b_{ij}}, \\ \omega_j &= \mu_j \theta_j, \end{aligned} \tag{6.5}$$

where I_i and O_i are the input data isze and output data size of task i, and b_{ij} denote the bandwidth for offloading task i onto device j.

The energy consumption for task i on the leased resource of device j can be modelled

as follows.

$$E_{ij}^{comp} = P_j^{active} T_{ij}^{comp} + \sigma \frac{I_i + O_i}{b_{ij}}, \tag{6.6}$$

where σ is the energy consumption rate of the wireless medium transmission, and it varies depending on the type of the wireless medium. Therefore, the overall cost of executing task i on the resource of device j is

$$C_{ij}^{comp} = \alpha T_{ij}^{comp} + \beta E_{ij}^{comp}, \tag{6.7}$$

where α, β are two weight factors that can be adjusted by the app providers. $\alpha + \beta = 1$.

Moreover, mobile users need to value the cost of tasks executing on their own devices before submitting an offloading requests. The local task execution cost can be modelled as follows.

$$\begin{aligned} T_{im}^{comp} &= \frac{c_i}{\omega_m}, \\ E_{im}^{comp} &= P_m^{active} T_{im}^{comp}, \\ C_{im}^{comp} &= \alpha T_{im}^{comp} + \beta E_{im}^{comp}. \end{aligned} \tag{6.8}$$

The subscript m is used as a sign of local execution. The time execution and energy consumption are similar to the offloading cost models, without the data transmission. The notations of the models are summarized in Table 6.1.

6.2.2 Problem Formulation

Traditionally, the resource allocation phase in the auction will decide the allocation results solely based on the bids from the buyers and sellers. That is, the seller who offers the highest bid in the reverse auction wins. However, in the proposed offloading market for HMC, the auctioneer also needs to consider the offloading benefits for each mobile device, i.e., conserving energy and reducing execution time. Therefore, we formulate the proposed offloading market problem as to maximize the overall revenues for all the participants in the auction, with the consideration of the offloading benefits and system constraints of HMC environment. The problem is formulated in the form of integer linear programming (ILP) as follows.

Given a set of offloading requests $r_i \in R$ and a set of services $s_j \in S$, the problem consists of two parts. The first part is how to allocate the requests to the available service offers so that each mobile device user can gain maximum revenue from sharing their redundant resources (from service providers' perspective), while the cost of executing

tasks in terms of time and energy are minimized (from task offloading perspective), since a mobile device user can be both a resource seller and a resource buyer for offloading. The second part of the problem is to decide the payment for the sellers.

We first define the binary decision variables for the allocation results. Let x_{ij} denote that request r_i is allocated to service offer s_j if $x_{ij} = 1$, otherwise if $x_{ij} = 0$. Let y_j denote that service offer s_j wins the auction if $y_j = 1$ and otherwise if $y_j = 0$. In order to ensure a valid task schedule for the machines, two supporting variables are introduced. Let t_{ij}^s be a continuous variable representing the execution starting time of the task i on service j, and o_{ikm} be a binary variable representing the order of task schedule, for which $o_{ikj} = 1$ if task i is in front of task k on the service j. Then t_{ij}^f can represents the finishing time of task i on service j, where $t_{ij}^f = (t_{ij}^s + T_{ij}^{comp})x_{ij}$. The problem is formulated as follows.

$$max \sum_{j \in S} y_j \sum_{i \in R} b_{ij} x_{ij} - \sum_{i \in R} \sum_{j \in S} C_{ij}^{comp} x_{ij} \tag{6.9}$$

s.t.

$$x_{ij} \leqslant y_j, \forall i \in R, \forall j \in S \tag{6.10}$$

$$\sum_{j \in S} x_{ij} = 1, \forall i \in R \tag{6.11}$$

$$x_{ij} C_{ij}^{comp} + \overline{T}(x_{ij} - 1) \leqslant C_{im}^{comp}, \forall i \in R, \forall j \in S \tag{6.12}$$

$$\sum_{i \in R} x_{ij} T_{ij}^{comp} \leqslant y_j T_j, \forall j \in S \tag{6.13}$$

$$t_{ij}^f \leqslant t_i^{arrival} + d_i, \forall i \in R, \forall j \in S \tag{6.14}$$

$$x_{ik} + x_{ki} + o_{ikj} + o_{kij} \leqslant 3, \forall i, k \in R, \forall j \in S \tag{6.15}$$

$$\overline{T}(o_{ikj} - 1) \leqslant t_{kj}^f - t_{ij}^s \leqslant \overline{T} o_{ikj}, \forall i, k \in R, \forall j \in S \tag{6.16}$$

$$t_{ij}^s \geqslant t_j^{join} + \overline{T}(x_{ij} - 1), \forall i \in R, \forall j \in S \tag{6.17}$$

$$t_{ij}^s \geqslant t_i^{arrival} + \overline{T}(x_{ij} - 1), \forall i \in R, \forall j \in S \tag{6.18}$$

$$t_{ij}^f \leqslant t_j^{leave} + \overline{T}(x_{ij} - 1), \forall i \in R, \forall j \in S \tag{6.19}$$

$$x_{ij}, y_j \in \{0, 1\}, \forall i \in R, \forall j \in S \tag{6.20}$$

The first component in the objective function (6.9) represents the overall payment received by service sellers on the market, which is the sum of the winning bids, and the second component in the objective function (6.9) represents the overall execution cost of the task requests on the market. Constraint (6.10) ensures that only the devices that have won the auction can be used for offloading. Constraint (6.11) shows that one task can be offloaded to one and only one device for execution, that is, the task is either offloaded

to a mobile device that offers its resources (device j), or executed on its own device i. Constraint (6.12) guarantees that the execution cost of the task on an offloaded device is less than running locally. Constraint (6.13) ensures the tasks allocated to the device is less than or equal to the device maximum available time for the offloading services. Constraints (6.14) prevent the task from finishing after its deadline. Constraints (6.15)-(6.19) ensure that there is one and only one task executing at a time on the service providing machines, and there is no overlapping on the execution orders. At last, Constraint (6.20) ensures the decision variables are either 0 or 1.

The optimal solution can be obtained by solving the proposed ILP model. However, the problem is NP-hard as it can be shown that the knapsack problem can be polynomially reduced to the proposed problem (6.9)-(6.20).

Theorem 6.1. *The proposed offloading market problem is NP-hard.*

Proof. We first show that a simplification of the proposed offloading market problem is NP-hard by giving a solution from reducing polynomially from the *Knapsack problem*. Then the simplification is relaxed to the original problem and prove it is NP-hard.

Assume that each mobile device on the market has a number of identical tasks to offload. Each mobile device as the resource seller has different processing capacity and a maximum service period limit. Therefore, the identical tasks may have different execution costs C_{ij}^{comp} on different sellers. Then this simplified offloading case can be seen as a multiple knapsack problem where the assigned tasks on a knapsack (mobile device) cannot exceed its maximum service time.

Now we relax the assumption that each mobile device can offload heterogeneous tasks. Since there are finite number of tasks from each mobile devices and a finite number of mobile devices on the offloading market, it takes polynomial time to calculate the execution cost of each task on each mobile device, and then devising the allocation of the set of offloading tasks onto mobile devices is corresponding to the simplified case. Therefore, the proposed offloading market problem (6.9)-(6.20) is NP-hard. □

We can observe that the term $y_j \sum_{i \in R} b_{ij} x_{ij}$ is the product of two binary variables, which is not a linear expression. This can be relaxed to a linear form by adding another binary variable and related constraints to replace the quadric term. The linearisation is

as follows. First, a binary variable z_{ij} is defined as

$$z_{ij} = x_{ij} * y_j. \tag{6.21}$$

Then the quadric term is replaced by the z_{ij} equivalently with the following additional constraints:

$$z_{ij} \leqslant x_{ij}, \tag{6.22}$$

$$z_{ij} \leqslant y_j, \tag{6.23}$$

$$z_{ij} \geqslant x_{ij} + y_j - 1. \tag{6.24}$$

Therefore, the integer linear programming formulation of the proposed problem is given as:

$$\begin{aligned} max : & \sum_{i \in R} \sum_{j \in S} (b_{ij} z_{ij} - C_{ij}^{comp} x_{ij}) \\ s.t. : & (6.9) - (6.24) \end{aligned} \tag{6.25}$$

The optimal allocation results can only be devised for the small-scale HMC network. It is too time-consuming for the auctioneer to obtain optimal solutions when there are a large number of mobile devices and offloading requests. Therefore, an online incentive mechanism is needed to solve this problem at runtime and produce the near-optimal solutions. Moreover, since the true cost and valuation of the offloading requests for the buyers and sellers are confidential to themselves only, the only information exposed to the auctioneer is the offloading request and the bidding price of the seller to fulfil the request. Hence, the incentive mechanism should guarantee a few economic properties of the auction to force both sides to report their true valuations. Based on the demand of the proposed offloading market for HMC, the online auction mechanism should fulfil the following properties [173].

- Computational efficiency: the auction outcome including winning sets of buyers and sellers, the mapping, and the clearing price and payment, are solved with a polynomial time complexity.

- Individual rationality: no person should lose money from joining the auction. In particular, let p_j be the payment for s_j set by the auctioneer, and b_j be the bid, $p_j \geqslant b_j$.

- Truthfulness: All players should use the strategy of only reporting his true valuation. No buyers or sellers can improve their utility by reporting a bid/ask different from its true valuation.

Therefore, we propose a greedy algorithm for resource allocation and pricing in the incentive mechanism to solve the ILP model and achieve near optimal results as well as the abovementioned economic properties.

6.3 The Incentive Mechanism

In this section, we present a greedy reverse auction algorithm that allocates the buyers' offloading requests with the sellers' service offers, and decides the payments for the successful transactions. The proposed algorithm takes into consideration the offloading benefits as well as the constraints in the HMC environment when allocating task offloading requests. Furthermore, the proofs of the economic properties of the proposed algorithm are presented.

6.3.1 The Greedy Reverse Auction

The online reverse auction in the proposed offloading market can be described as a series of auctions on a timeline that consists of discrete time epochs. Sellers offers s_i and the buyer offloading requests r_i arrive at arbitrary times. For each s_i, it publishes the service information including its service type, capacity, bid, and available period to the auctioneer. The published information will be discarded after its available period. For each request r_i, it contains the requested amount of computation and the user nominated task completion deadline. In each auction time epoch, the greedy reverse suction algorithm performs two phases: resource allocation, and payment determination.

In the resource allocation phase, although there may be more than one offloading request submitted to the auctioneer in the current auction epoch, it rarely happens when two offloading requests arrive at the auctioneer at the exact same time. Therefore, all the submitted offloading requests are put in a queue based on the order of arrival. The greedy reverse auction algorithm processes the requests in the queue one per time to allocate to the available service offers based on the task requirement, offloading benefits and environment constraints in HMC. The details of the proposed resource allocation algorithm are listed in Algorithm 6. The algorithm takes the set of offloading requests R and the set of service offers S as inputs, and returns the values of the decision vari-

Algorithm 6 Resource Allocation Algorithm

1: **procedure** RESALLOC(R, S)
2: Initialize $R_{sort}, S_{sort}, x_{ij}, y_j$;
3: **for all** $r_i \in R$ **do**
4: $R_{sort} \leftarrow Sort(R_{sort}, r_i, c_i, d_i, "descend")$;
5: **for all** $s_j \in S$ **do**
6: $S_{sort} \leftarrow Sort(S_{sort}, s_j, \frac{b_j}{\omega_j}, "ascend")$;
7: **for all** $r_i \in R_{sort}$ **do**
8: **for all** $s_j \in S_{sort}$ **do**
9: **if** $\frac{c_i}{\omega_j} \leqslant d_i \wedge \frac{c_i}{\omega_j} \leqslant T_j \wedge C_{ij}^{comp} \leqslant C_{im}^{comp}$ **then**
10: **if** $y_j \neq 1$ **then**
11: $s_j \leftarrow update(T_j, \frac{c_i}{\omega_j})$;
12: $y_j = 1$;
13: $x_{ij} = 1$;
14: break;

ables x_{ij}, y_j as outputs. The algorithm first sorts the queue of offloading requests in the descending order of the value of $\frac{c_i}{d_i}$, which reflects the priority of the requests in terms of computation amount and deadline of the task (step 3-4). That is, the request with large computation or urgent deadline will be processed first. Similarly, the service offers are also sorted into the list S_{sort} in the ascending order based on the value of $\frac{b_j}{\omega_j}$ (step 5-6) so that the service with lower bid and higher processing speed will be placed in front. Then the algorithm takes the sorted requests and offers to start the auction process (step 7-14). For each request r_i at the front of the queue R_{sort}, the algorithm iterates through S_{sort} to find the winners when three conditions are satisfied (step 8-14): 1) the offloading request can be completed before its deadline ($\frac{c_i}{\omega_j} \leqslant d_i$); 2)the service can finish the offloading request before its available time ends ($\frac{c_i}{\omega_j} \leqslant T_j$); 3)the offloading benefit for request r_i is ensured ($C_{ij}^{comp} \leqslant C_{im}^{comp}$). The winner y_j will be assigned to the request x_{ij} and considered unavailable to other requests in this auction epoch. The available time T_j of service s_j is updated substituting the execution time of r_i (step 11). At last, the algorithm returns the results of x_{ij}, y_j after all the offloading requests are processed.

In the second phase of the greedy reverse auction algorithm, the payments of the winners are determined. Traditionally, the payment can be determined by calculating the difference of the optimal result of the objective function with and without the participation of the winners. Mathematically, let F_{S-j} denote the optimal result of the objective

Algorithm 7 Payment Determination

1: **procedure** PAYDET(R, S_{sort})
2: $s^* \leftarrow critical(S_{sort})$
3: $S_{win} \leftarrow winnerSelect(S_{sort})$
4: **for all** $s_j \in S_{win}$ **do**
5: $p_j \leftarrow \frac{b^*}{\omega^*}\omega_j$

Algorithm 8 Greedy Reverse Auction Algorithm

1: **procedure** GREEDYAUCTION(D)
2: **for all** $d_k \in D$ **do**
3: $R \leftarrow$ current available offloading requests
4: $S \leftarrow$ current available resource sellers
5: $x_{ij}^{d_k}, y_{ij}^{d_k}, S_{sort} \leftarrow ResAlloc(R, S)$
6: $p_j^{d_k} \leftarrow PayDet(R, S_{sort})$

function without the presence of service j, and F_S^{-j} denote the optimal result of the objective function without considering the bid of service j, given j as the winner in the optimal allocation results of the formulation. Then the payment is calculated as:

$$p_j = F_{S-j} - F_S^{-j}. \tag{6.26}$$

Since the obtaining optimal results of the objective function is not feasible at runtime, we proposed the payment algorithm based on the allocation results of the greedy auction algorithm. Similar to the optimal payment policy, the payment to the winner is the difference of the overall system utility with and without the presence of the winner. After all the offloading requests have been allocated to services in S by the greedy auction algorithm, the next unallocated service offer s^* in the sorted list S_{sort} is selected as the *critical service*. If all the service offers are winners, then the last winner is selected as the *critical service*. The *critical service* represents the difference of overall system utility when taking out one of the winners in S_{sort}, due to the fact that since the service offers are sorted in the ascending order of the bid, the winners are in front of s^* in S_{sort}, and when taking out one of the winners the *critical service* s^* will be the winner of the original auction. The bid of the *critical service* is called the *critical price* b^*. Then the payment to winner s_j can be calculated as $p_j = \frac{b^*}{\omega^*}\omega_j$. The payment determination algorithm is listed in Algorithm 7.

Assuming there are m offloading requests and n service offers, the computation complexity of the proposed greedy reverse auction algorithm in one auction time epoch is $O(mn(m \log m + n \log n))$, where the lowest complexity of sorting of the request set and

offer set are $O(m \log m)$ and $n \log n$ respectively. Then assuming for the worst case, allocate one request to the service cost n operations, then m iterations cost mn operations. Therefore the overall computation complexity of the proposed algorithm is $O(mn(m \log m + n \log n))$. The proposed greedy reverse auction algorithm in the scale of a series of D auction time epochs is listed in Algorithm 8. In each auction time epoch d_k, the algorithm collects the available offloading requests and service offers from the mobile users on the offloading market first. Then the allocation results and payments are determined by the Algorithm 6 and Algorithm 7 respectively. The algorithm returns the overall allocation results X_{ij}^D, Y_j^D and payments P_j^D after $|D|$ time epochs.

6.3.2 Failure Recovery and Penalty Policy

Since the mobile devices on the offloading market can join and leave at any time, the failure penalty policy is needed to prevent mobile device users who win an auction from deliberately failing the requests. We propose a penalty policy that only utilizes the available resources on the market and the support of the application provider.

When a mobile user who won the offloading request leaves the market and can not complete the request, the current state of the offloading task will be checkpointed [121] and is sent to the cloud instance provided by the application provider. The remaining of the failed task will be completed on the cloud instance and the application provider charges the cost of this execution to the leaving mobile device user as a penalty, and the payment for the leaving user is reversed. In this way, the offloading user will lose nothing and have the offloading request completed, while the leaving user will pay the cost of execution as a penalty.

6.3.3 Economic Property Analysis

Having proposed the reverse auction based incentive mechanism, we prove that the proposed incentive mechanism satisfies the economic properties of individual rationality and truthfulness. These properties guarantee that the participants on the offloading market will not benefit from cheating, and avoid the manipulation of the offloading market.

Theorem 6.2. *Truthfulness: the greedy reverse auction algorithm implements a truthful auction where no participants can increase its utility by bidding a price different from its private true valuation.*

Proof. Since the true valuation of the resource provided by the sell is private, the incentive mechanism should guarantee sellers to report their true valuation so that they will not cheat to gain more by bidding a higher price than their true costs. Let b_j be the bid of seller s_j, v_j be the true valuation of the seller's cost, and u_{b_j}, u_{v_j} be the utility of the two biddings respectively. There are two cases need to be considered, which are $b_j < v_j$ and $b_j > v_j$.

First, when $b_j < v_j$, the outcomes of the auction for seller s_j can be: (A)s_j loses the auction by bidding v_j and b_j; (B)s_j wins by bidding b_j but loses by bidding v_j; and (C) s_j wins by bidding either b_j or v_j. Note that the case where s_j wins by bidding v_j and loses with bid b_j does not exist since the sellers' offers are sorted in the ascending order of bids in Algorithm 6.

For case (A), the utilities of seller s_j by bidding b_j and v_j are both equal to 0 since the requests are allocated to the sellers with cheaper bids. That is, $u_{b_j} = u_{v_j} = 0$.

For case (B), based on the payment policy which finds the critical price $b*$ among all sellers' bids, there exists $\frac{b_j}{\omega_j} < \frac{b^*}{\omega^*} < \frac{v_j}{\omega_j}$. Since the utility of the winner is calculated as $u_{b_j} = p_j - v_j$, the winner s_j in this case would obtain a negative utility. Therefore, the seller s_j would choose to either lose the utility or lose the auction, which results as $u_{b_j} \leqslant 0$.

For case (C), since the sellers' offers are sorted in the ascending order of the bids, the seller would be placed closer to the from of the list S_{sort} when bidding with b_j than v_j. However, since this has no effect of the value of the critical price b^*, the utility of $u_{b_j} = u_{v_j} = b^* - v_j$.

As a result, for all the possible auction outcomes when the seller s_j bids lower than his private true valuation, the utility is less or equal to the utility of bidding the true valuation, i.e., $u_{b_j} \leqslant u_{v_j}$. It can be proved in a similar way that $u_{b_j} \leqslant u_{v_j}$ when $b_j > v_j$. Therefore, the proposed greedy reverse auction algorithm can guarantee the truthfulness of the participants. □

Theorem 6.3. *Individual Rationality: The proposed greedy reverse auction algorithm is individual rational that no participants would lose money from joining the offloading market.*

Proof. In order to prove the individual rationality of the proposed algorithm, we need to prove that $p_j \geqslant b_j$ for all the winners $s_j \in S$. According to the Algorithm 7, each winner will be paid with the value of $\frac{b^*}{\omega^*}\omega_j$, which is based on the bid of the first unallocated seller or the last seller if all sellers are allocated. Recall that all the sellers are sorted in the ascending order of $\frac{b_j}{\omega_j}$ in Algorithm 6, then for each winner s_j we have

$$\begin{aligned} p_i &= \frac{b^*}{\omega^*}\omega_j \\ &\geqslant \frac{b_j}{\omega_j}\omega_j = b_j \end{aligned} \tag{6.27}$$

Therefore, for any winner in the auction, the winner's payment $p_j \geqslant b_j$, and the proof is complete. □

6.4 Performance Evaluation

In this section, we present the performance evaluation of the proposed greedy reverse auction algorithm. The experiments are discussed in two parts. First, the proposed algorithm is evaluated under intensive simulations to test the performance in terms of the social welfare, offloading benefits, the fairness of the auction results, and the truthfulness and individual rationality enforced by the algorithm. Second, we implement the prototype of the incentive mechanism on the Android operating system and test the feasibility of the mechanism in real setting experiments. For the rest of this section, we refer to our proposed incentive mechanism as *mCloudAuc*.

6.4.1 Simulation Settings

We conduct the simulations based on a real-world trace of mobile user activities and mobile application profiles. The main parameters for the simulations are listed in the Table 6.2.

First, we profile an OCR application[2] on an Android mobile phone to generate the task workloads for the simulations. The Android Studio performance analysis tools are

[2]Available to download at https://github.com/rmtheis/android-ocr

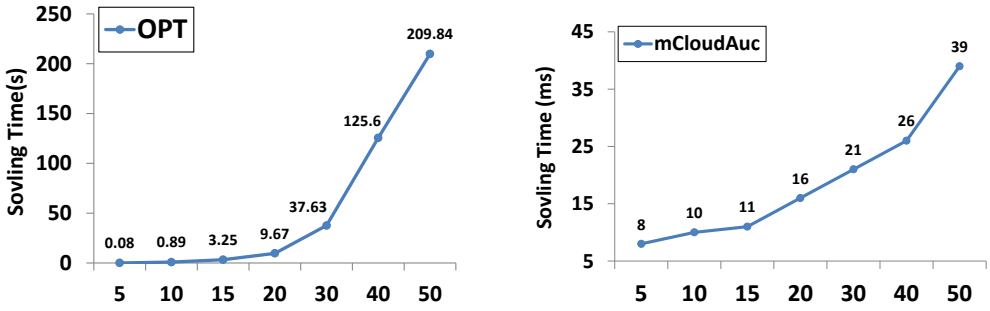
Table 6.2: The configuration of simulations

Workload	**Application:** Optical Character Recognition **Task arrival rate:** 0.5 **Computation required (Instructions):** Uniform(50000, 100000) **Input data size (MB):** Uniform(0.5, 10)
Mobile device	**Behaviour trace:** MIT Reality Mining traces **Processing Speed (MIPS):** Uniform(5000,200000) **Energy consumption rate (W):** Uniform(0.07,0.6) **WiFi bandwidth(MBps):** Uniform(1.0,1.2) **Bluetooth bandwidth(MBps):** Uniform(0.2,0.3) **Cellular bandwidth(MBps):** Uniform(0.8,0.9) **Energy consumption rate of wireless medium:** $\rho_{wifi} = 1.94W, \rho_{bt} = 0.28W, \rho_{cell} = 5.56W$

used for the profiling of the application on the method level. The profile contains the input data size of the captured photos, and the inclusive CPU running time of each method for offloading to calculate the amount of computation in instructions. The task arriving rate λ at the auctioneer is 0.5.

Second, the dataset MIT Reality Mining [62] is used to extract the mobile device behaviour trace for our experiments. The dataset contains the trace of 100 mobile devices in the MIT Media Lab, which consists of the location and staying duration, other devices in its proximity, idle/charging state, etc. We only consider the logs related to the cell tower associated with 'Work'. The list of such cell towers is provided by the project[61]. We first filter the datasets with the cell tower IDs to find devices connected to each of the cell towers and the corresponding connection time. Then for each set of devices found connected to each cell tower, the groups of mobile devices in each other's proximity are identified, and each group is assigned with a unique group ID. Last, the state including charge, active, and on/off are retrieved from the datasets for each device in the found groups.

In addition, the hardware properties such as processing speed and energy consumption rates are appended to the trace. Million Instructions per Second (MIPS) is used to quantify the processing speed, and is generated from a uniform distribution of $(5000, 200000)$. The energy consumption rates of mobile devices are obtained from the uniform distribution of $(0.07, 0.6)W$ corresponding to the processing speed of the processor. The power



(a) Optimal solution (b) mCloudAuc

Figure 6.2: Time of devising allocation results with different number of machines

consumption rate of WiFi, Bluetooth, and cellular network are set as $\rho_{wifi} = 1.94W, \rho_{bt} = 0.28W, \rho_{cell} = 5.56W$ respectively [15]. Finally, the network speed ($MBps$) of WiFi, Bluetooth and cellular network are set as $B_{wifi} = Uniform(1.0, 1.2), B_{bt} = Uniform(0.2, 0.3), B_{cell} = Uniform(0.8, 0.9)$ respectively by profiling the available network in our experiment environment.

6.4.2 Numerical Results and Analysis

We conduct 5 sets of experiments for simulation to evaluate the economic properties and the performance in terms of offloading benefits for the proposed incentive mechanism.

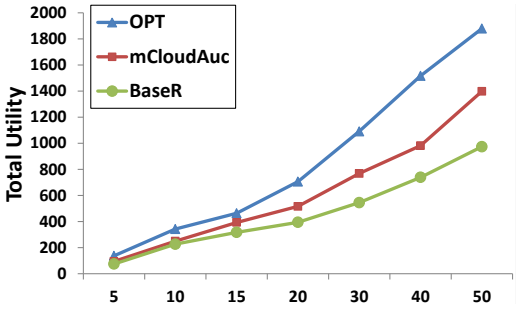
Running Time Analysis

First, we the running time of obtaining the solutions of the optimal ILP model and the proposed greedy reverse auction algorithm are compared for a different number of mobile devices on the offloading market. The optimal solution of the ILP model is obtained by using the Gurobi optimization solver [168]. The running time of generating the allocation solution for the optimal algorithm and the proposed greedy auction algorithm are compared against different a number of mobile devices on the offloading market. The generating rate of offloading task requests for each device is randomly drawn from the uniform distribution$(0.5, 2)$. The number of task requests in each test run is managed to be around 100. All the tests are conducted on a desktop with 3.40GHz Intel Core i7 processor and 16 GB RAM. The results of the solving time with different numbers of machines are depicted in Figure 6.2.

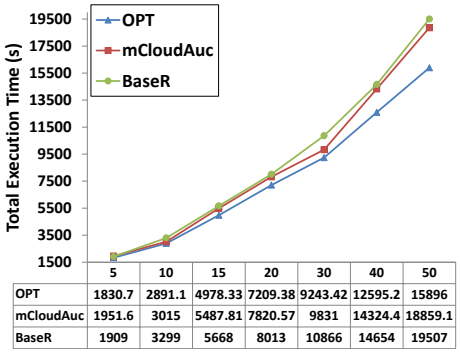
The first observation of the difference between the optimal solution and the proposed mCloudAuc algorithm in Figure 6.2 is that the optimal solution takes a significantly more amount of time to devise allocation results for the auction, as it only takes below 40 milliseconds for mCloudAuc (Figure 6.2a) to obtain results while the optimal solutions took from 80 milliseconds up to 209 seconds (Figure 6.2b). The running time for both algorithms increase with more mobile devices participating, and the optimal solution increases more rapidly. These results indicate a useful setting for the auction system configuration. Since the auction is designed to conduct on the cloudlet in the offloading market (Figure 6.1), the service providers can set up a threshold of the number of mobile devices based on their SLAs. The auctioneer can apply optimal solution when the actual number of participants is below the threshold, or the mCloudAuc algorithm when it is above the threshold.

Economic property evaluation

In the next three sub-sections, the performance of the proposed auction algorithm is evaluated regarding the three economic properties: social welfare, individual rationality, and truthfulness. Three algorithms are compared, including 1) the optimal solution from ILP model as the benchmark, referred as **OPT**; 2) a reverse auction based incentive algorithm proposed in [241] referred as **BaseR**; 3) and the proposed incentive algorithm referred as **mCloudAuc**. The BaseR algorithm sort bidders in a non-decreasing order of $\frac{\omega_i}{d_i}$, where ω_i represents the valuation of the task for each bidder, and d_i represents the degree of the bidder in its current computing network. Since the network in our case can be considered as a complete graph, the BaseR is adapted to sort by the value of ω. The adapted BaseR algorithm sorts the bidders at once at the beginning, and the order will be fixed. The tasks are assigned on arrival with a winner from the sorted list of bidders. Firstly, the performance of the proposed mCloudAuc against the benchmark and baselines in terms of the overall utility of all participants are analysed with different task arrival rates and different numbers of devices on the market. The utility refers to the revenue that a participant receives. In addition, the overall task execution time and energy consumption of all the submitted tasks are compared. The task generating rate for each mobile device is set to follow a Poisson process with $\lambda = 0.5$. The results of each test are av-

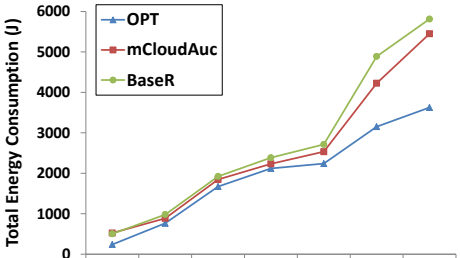


(a) Overall utility



	5	10	15	20	30	40	50
OPT	1830.7	2891.1	4978.33	7209.38	9243.42	12595.2	15896
mCloudAuc	1951.6	3015	5487.81	7820.57	9831	14324.4	18859.1
BaseR	1909	3299	5668	8013	10866	14654	19507

(b) Overall task execution Time



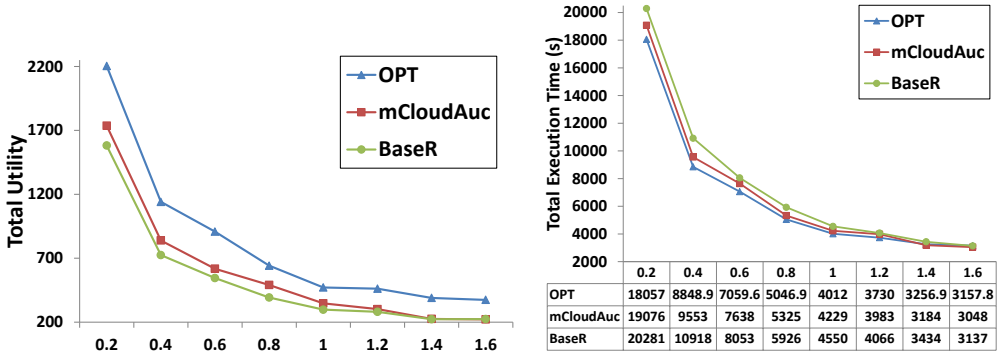
	5	10	15	20	30	40	50
OPT	240.34	764.18	1671.27	2121.2	2239.89	3152.97	3624.8
mCloudAuc	524	885.84	1845.9	2233	2534	4225	5448.26
BaseR	504	980	1921.73	2386	2713	4888	5813

(c) Overall device energy consumption

Figure 6.3: Overall utility with different number of participants

eraged by 50 runs, and depicted in Figure 6.3. Figure 6.3a shows the sum of the utility for each participating mobile device on the offloading market. In comparison with the other algorithms, mCloudAuc generates close results to the optimal algorithm, and outperforms the well-designed BaseR algorithm when the number of participants grows. In terms of task offloading benefits such as task execution and energy conservation, Figure 6.3b and Figure 6.3c show that all three algorithms perform closely to each other, while mCloudAuc can conserve more energy and execution time compared to BaseR. This is due to the dynamic ordering of both task requests and resource seller offers during each round of the auction in mCloudAuc, whereas BaseR applies a fixed order of offers when having auctions.

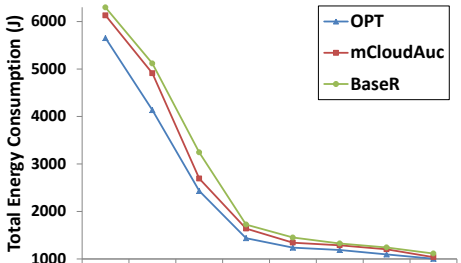
Moreover, the performance of overall utility for the algorithms is tested with different task arrival rates (or task request submission rates) for the mobile participants. Since the task arrival follows the Poisson process in our simulation, the λ is set from 0.2 up to 1.6, and eight sets of simulation are conducted to compare the overall utility, total execution



(a) Overall utility

	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6
OPT	18057	8848.9	7059.6	5046.9	4012	3730	3256.9	3157.8
mCloudAuc	19076	9553	7638	5325	4229	3983	3184	3048
BaseR	20281	10918	8053	5926	4550	4066	3434	3137

(b) Overall task execution Time



	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6
OPT	5650.5	4135.5	2432.1	1436.5	1237.8	1188.7	1095	1004.1
mCloudAuc	6129	4911	2692	1639	1343	1287	1204	1037
BaseR	6295	5117	3243	1723	1451	1326	1242	1114

(c) Overall device energy consumption

Figure 6.4: Overall utility with different task arrival rates

time and device energy consumption. The workload for each test case is obtained by generating the task requests up to from 0 to 100 second of the arrival time. The results are averaged with 50 runs of each set of test and are illustrated in Figure 6.4. As seen in Figure 6.4a, the total utility of mobile participants decreases as the task arrival rate decreases from 0.2 to 1.6, due to fewer task requests submitted during the 100-second auction window. Noticeably, the utility generated by mCloudAuc when the arrival rate is greater than one is as close as that of the OPT. The result indicates that the proposed mCloudAuc can give near-optimal allocation results when the task arrival rate is low ($\geqslant 1$). The corresponding execution time and device energy consumption (Figure 6.4b and 6.4c) are consistent with the utility results. Three algorithms all generate close results, with the proposed mCloudAuc slightly outperforms BaseR. The two sets of simulation conducted above show that the proposed incentive algorithm can generate resource allocation that is close to the optimal solution and maintain the offloading benefits of mobile cloud offloading at the same time.

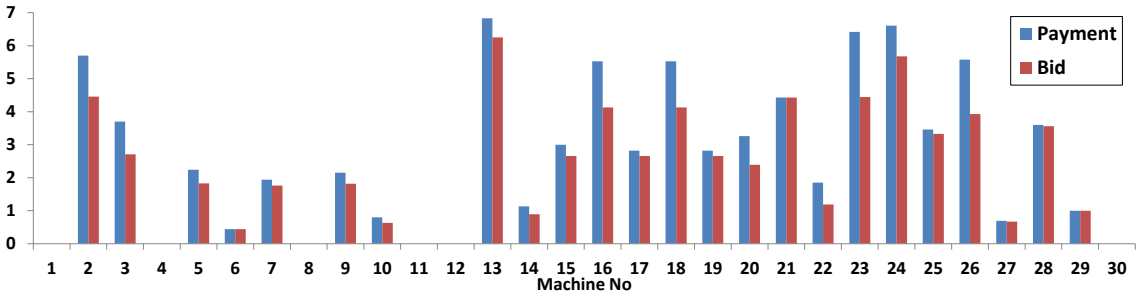
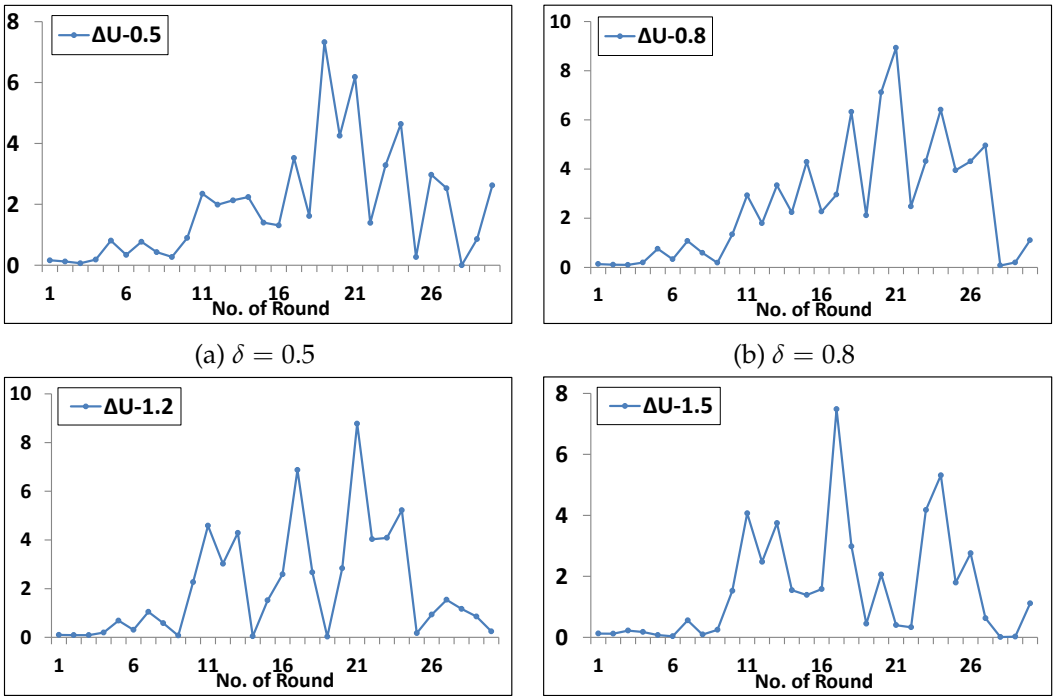


Figure 6.5: Individual rationality of mCloudAuc

Secondly, the price determination algorithm of mCloudAuc is evaluated to valid the individual rationality of the proposed incentive algorithm. The individual rationality of an auction mechanism ensures that participants of all parties would not lose profit from joining the auctions. Since the designed offloading market implements a reverse auction, the evaluation only focuses on the resource sellers' profits. A workload of 100 task requests randomly generated from the mobile device participants on the offloading market is tested. The bid and price paid for each resource seller and buyer pair are compared. In case a seller wins multiple task requests from different buyers, the sum of the bids and price paid are compared. The results are averaged with 50 runs, and are shown in Figure 6.5. The results with 0 (e.g., machine 1,4,8) indicate that the mobile device has not won any task requests. As can be observed in Figure 6.5, all the mobile devices (or resource sellers) receive payments are at least no less than what their bids are. Therefore, the proposed mCloudAuc incentive mechanism can achieve the individual rationality as the Theorem 6.3 shown.

Finally, we evaluate the truthfulness of the proposed mCloudAuc algorithm. A brief remind that truthfulness of an auction mechanism enforces participants to only bid on their true valuation of the tasks. That is, no participants would gain more profits by bidding more than their true cost. In order to test the truthfulness performance of mCloudAuc, a time-based simulation with several rounds of auctions held in between is conducted to test the utility changes of the participants. Mobile device participants may join and leave at each round. Instead of generating from the distributions in Table 6.2, the participant's joining and leaving time used in this test are extracted from a part of the MIT Reality Mining traces. The workload consists of task requests submitted from the participants over the rounds of auctions, following a Poisson process with $\lambda = Uniform(0.2, 1)$. One



(a) $\delta = 0.5$ (b) $\delta = 0.8$

(c) $\delta = 1.2$ (d) $\delta = 1.5$

Figure 6.6: Utility loss (ΔU) with different bid manipulation factor δ

mobile device participant is randomly chosen as the reference device. In each round of auctions, the selected device is manipulated to bid untruthfully from its true valuations, that is, $b_i = \delta c_i$, where δ is a weight factor to control the variation. Then the utility loss $\Delta U = U_{mCloudAuc} - U_{mCloudAuc-M}$ generated by mCloudAuc between manipulated settings $mCloudAuc - M$ and original settings $mCloudAuc$ is depicted in Figure 6.6, with different values of δ including 0.5 (Figure (6.6a)), 0.8 (Figure (6.6b)), 1.2 (Figure (6.6c)), and 1.5 (Figure (6.6d)), which represent the selected seller underbidding his or her true cost or overbidding. Each auction epoch is set as 30 seconds.

All four subfigures show non-negative utility loss under different manipulation factor δ. The different variations for the four cases are due to the resource allocation policy and price determination policy proposed in mCloudAuc, in which the task requests and service bids are sorted at each auction round, and the payment price (or critical price) changes when the order of service bids changes with untruthful bids. The results indicate that the proposed price determination algorithm in mCloudAuc can ensure truthfulness for the participants during auctions when there are malicious participants try to gain

extra utility by underbidding or overbidding their true private valuations.

The above sub-sections evaluates the computation efficiency and economic properties of the proposed incentive mechanism mCloudAuc by simulations. The results show that mCloudAuc performs consistently with Theorem 6.2 and Theorem 6.3, and can provide near-optimal performance in terms of utility and mobile offloading benefits.

The Implementation of mCloudAuc on Android Operating System

In order to test the feasibility of the proposed mCloudAuc mechanism on real applications, we implement a prototype of the incentive mechanism framework on the Android platform (Android 5.0 with API level 21). The framework is designed in a server-client architecture where the client layer runs in Android applications, and the auctioneer server runs on a cloudlet in an HMC network (as shown in Figure 6.1). In this section, the design and interaction of the modules in the framework are explained first for the client and the server, then a set of experiments are conducted on the implemented prototype using a mobile OCR application to evaluate the response delay of the proposed mCloudAuc framework running with real mobile applications. Figure 6.7 illustrates an example of one auction process and the related interactions between modules on the client and the server.

The client side of the framework is implemented on the Android platform. The client framework provides the functions including a *Communication* module, an *Offloading Handler*, and a *Resource Sharing Handler*. The *Communication* module runs in a thread to deal with data encoding and decoding, commands exchange and handling with the auction server. The *Offloading Handler* offloads mobile tasks and receives results. The *Resource Sharing Handler* registers the service offers with bids and executes the offloaded tasks from other mobile users on the shared resources.

On the auction server side of the mCloudAuc framework, five modules are implemented, including a *Communication* module, a task offloading request registry *TaskRegistry*, a service offer registry *ServiceRegistry*, a *Resource Allocator*, and a *Balance Database*. Similar as the client, the *Communication* module handles the data transmission and commands sent from the connected mobile devices, and it runs in a separate thread. The *ServiceRegistry* and *TaskRegistry* maintain the currently available service offers and task

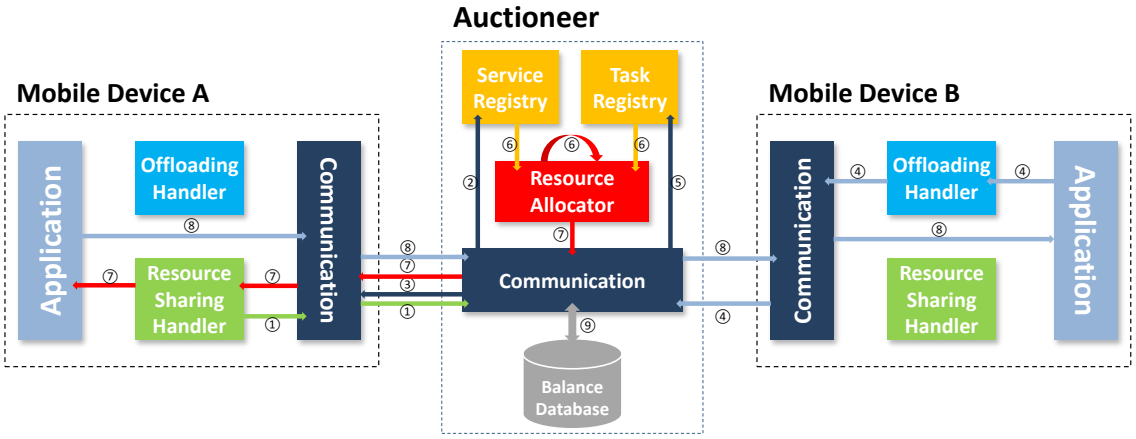
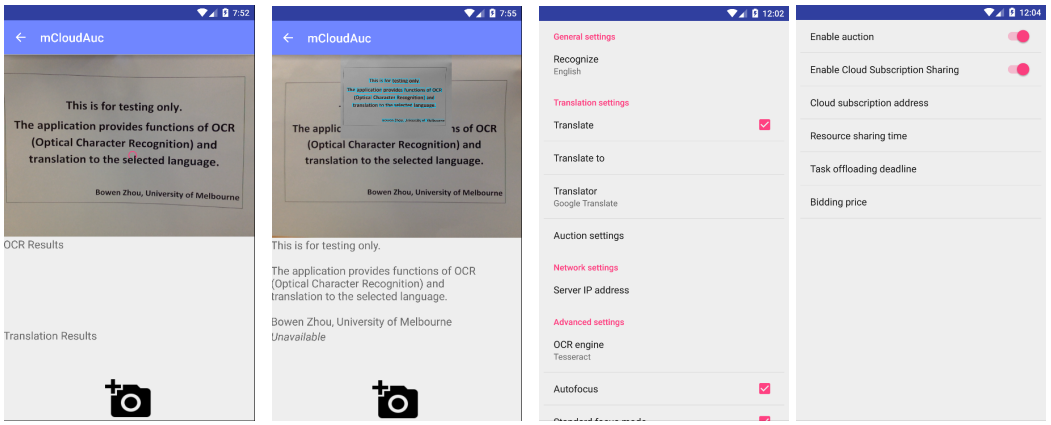


Figure 6.7: An example of one auciton process and interactions of modules in the framework

offloading requests respectively. The *Resource Allocator* implements the proposed auction logic to assign task requests to the service offers and proceeds the payments. The *Balance Database* is implemented with the NoSQL database MongoDB to provide a fast and lightweight data storage solution for managing the credit balance for users registered with the auctioneer. The example in Figure 6.7 shows the interactions of modules on mobile devices and the auctioneer, where the auctioneer allocates a task offloading request from the buyer *Device B* to a seller *Device A*. ①: *Device A* sends a service offer registration {SERVICE_ REGISTRATION,offer} through its *Resource Sharing Handler* to the *Auctioneer* with provided processing capacity, available time and the bidding price. ②: the service offer is added to *Service Registry*. ③: *Auctioneer* sends back the confirmation with the assigned UUID {REGISTRATION_SUCCESS, service_UUID} to *Device A*. ④: *Device B* submits a task offloading request {SUBMIT_ TASK_ REQUEST, task} with the task and deadline through its *Offloading Handler*. ⑤: the *Auctioneer* adds the offloading request to its *Task Registry*. ⑥: When each round of the auction starts, the *Resource Allocator* takes in the list of available services and task requests to allocate the tasks using the proposed auction algorithm. ⑦: After *Auctioneer* makes the decision to allocate task request from *Device B* to service offer provided by *Device A*, it sends the task {OFFLOAD_TASK, task} to *Device A*. ⑧: The application on *Device A* executes the offloaded task as an AsyncTask at background and sends the execution results {SUBMIT_RESULT, result} to *Device B*. If Device A is sharing its cloud subscription, the task will be sent to its dedicated instance on the cloud to execute. ⑨: The *Communication* module periodically stores the payment



(a) User interface (b) OCR results (c) Application settings (d) Auction settings

Figure 6.8: The user interface of the implemented mobile translator application

and balance information to the *Balance Database*. We implement the testing application with the framework based on an open-source Android OCR (Optical Character Recognition) application[3]. Figure 6.8 shows the user interface and application settings of the application. The user is able to change the settings that are related to the OCR, translation, and auction offloading in application settings. The experiment is conducted with three Android devices (one Nexus 5, one HTC EVO 3D, and one Samsung I9000) running the mobile translator application, a laptop running the auctioneer server, and an Amazon EC2 m4.xlarge instance running Genymotion on-demand Android 5.1 image as the cloud subscription. The translation application is set up to run at the background with a server socket open to listen for task offloading execution in the instance. The response delay of the auction processing time and the offloaded OCR + translation task execution time are measured using the implemented application to valid the feasibility of the proposed incentive mechanism in the real execution environment. In order to keep the results constant, we run the application on HTC EVO 3D with the same set of 10 photos and obtain averaged results of 50 runs. The results of response time and auction process time are shown in Figure 6.9. Note that since different photos have different OCR processing time based on the photo quality and text content, the results in Figure 6.9 are not comparable photo-wise. In Figure 6.9, the blue bar one the left of each photo item represents the local execution time of the image OCR and translation, the green bar on the right represents

[3]Available at: https://github.com/rmtheis/android-ocr

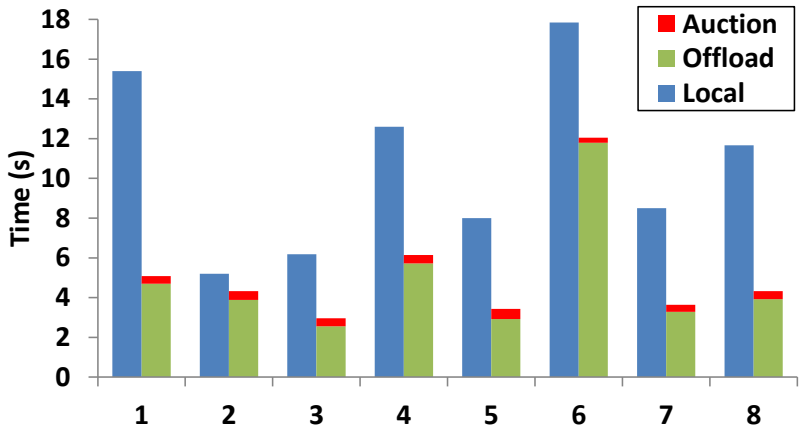


Figure 6.9: Comparison of task response time between local execution and using the mobile cloud offloading service

the execution time of the image OCR and translation when using the offloading service, and the red bar represents the processing time of communication plus the auction processing from the auction server. As can be observed from the figure, the auction process only takes around 0.3 second, which accounts for less than 10% of the whole task response time of the offloaded tasks. Moreover, the task response time for the image OCR is reduced by over 50% on average when using the offloading service, and the additional processing time of the auction process is negligible compared to the task response time. The experiments using the implemented mCloudAuc system demonstrate that the proposed auction process added to the offloading service is short for the whole application OCR processing (less than 300 ms), and can provide significant performance enhancement to the application task response time. Therefore, the prototype implementation of mCloudAuc shows the feasibility in practice.

6.5 Summary

Since the mobile cloud offloading services in HMC work in the form of opportunistic mobile network, the mobile device users may lack incentive to share their own resources due to the battery lifetime concern. In order to encourage users to commit in the offloading services, we propose an incentive mechanism in this chapter. The problem is formulated as an offloading market auction problem using ILP, where a mobile user who uses the mobile cloud offloading service can be a seller or a buyer, or both at the same time. A

seller refers to a mobile user who wishes to share the redundant resources for others to offload, and a buyer refers to a mobile user who requests remote resources for offloading his or her tasks. The sellers compete by bidding different prices for the task requests on the market, and receive payments from the buyers.

We proposed a reverse auction-based mechanism that includes an online resource allocation algorithm and a payment determination algorithm. The resource allocation algorithm schedules the offloading task requests with the available bids based on the offloading benefits and constraints in the HMC environment. We demonstrate the computation efficiency, individual rationality and truthfulness of the proposed algorithm by both theoretical proof and simulations. The simulation results show that the proposed algorithm has a near-optimal performance compared to the results obtained from the ILP models. We also implement a prototype system of the proposed incentive mechanism on top of our mCloud framework on the Android platform with a mobile OCR translation application to show its feasibility in practice.

Chapter 7
Conclusions and Future Directions

This chapter summarises the research works on enabling efficient and seamless task offloading services in the heterogeneous mobile cloud environment in this thesis. In addition, this chapter identifies some future directions to pursue in this area.

7.1 Conclusions and Discussion

MOBILE cloud computing has gained popularity in both industries and academia in recent years. As the three-tier cloud computing service model can provide layered, elastic, pay-as-you-go computing and storage services, it enables the resource-constrained mobile devices to utilize the cloud computing services to enhance its battery lifetime and mobile application performances. Task offloading is a major approach to bring the power of cloud computing to the mobile device's proximity. It provides solutions to identify the computation intensive tasks in a mobile application and migrate them to cloud resources for remote execution so that the battery of mobile devices can be conserved and the application performance is enhanced.

However, this conventional mobile-to-cloud offloading service model has a significant performance bottleneck caused by the wireless network condition, which affects the time needed for the task migration. As the recent development of hardware on smart mobile devices with advanced processor and sensors, the mobile device itself can also be a potential resource provider for the mobile task offloading service. Therefore, in this thesis, we proposed a new mobile cloud computing offloading service environment, the heterogeneous mobile cloud (HMC). In the HMC environment, public and private cloud

This chapter is partially derived from: **Bowen Zhou** and Rajkumar Buyya, "Augmentation Techniques for Mobile Cloud Computing: A Taxonomy, Survey, and Future Directions," *ACM Computing Surveys*, Volume 51, No. 1, Article No. 13, Pages: 1-38, ISSN 0360-0300, ACM Press, New York, USA, January 2018.

instances, cloudlets, and an ad-hoc network of smart mobile devices in each other's proximity form a resource sharing pool, in which each device can be a resource provider and a consumer to offload tasks at the same time. The new service environment creates a few new challenges, e.g. lightweight task offloading and scheduling algorithms, adaptive fault tolerant strategies, etc. that the traditional distributed system solutions have not considered. Chapter 1 particularly discussed the concepts, challenges, and the main contributions of this thesis to solve these new challenges.

In this thesis, the challenges of providing seamless HMC offloading services are classified and studied with two main aspects: 1) a cross-platform task offloading enabling technique for the defined heterogeneous mobile cloud (HMC) environment, and the lightweight task scheduling algorithms that can generate competitive schedules for offloading tasks among the hybrid cloud resources; 2) the supporting techniques to ensure the HMC system can provide reliable services and guarantee participants to gain benefits from participating.

Firstly, Chapter 2 conducted a literature review based on the above-mentioned challenges. A taxonomy and related survey are presented to investigate the state-of-the-art techniques adopted in the mobile cloud augmentation in terms of computation and mobile storage. In the first part, for the computation augmentation, related techniques that have been adopted in the literature are discussed, including the details of the technology, representative existing projects, and its advantages and disadvantages. In addition, the supporting techniques including context monitoring, decision making, and fault tolerance are also presented. In the second part of the literature review, frameworks and approaches for mobile cloud storage augmentation are discussed and compared.

In Chapter 3, a mobile task offloading service framework was proposed to enable the offloading service for the HMC environment. Modules and interactions between each module are explained to show the workflow of the system. A detailed discussion of the offloading technology and examples of the usage are presented. Moreover, a context-aware offloading decision making approach based on multi-criteria decision making method was proposed to decide whether to offload a mobile task based on the context changes of the execution environment in order to save battery and accelerate the task execution.

Chapter 4 took a further step to investigate the optimization of overall minimized application response time for all the mobile users within the HMC environment, subject to the context constraints of the HMC such as battery limit, device mobility, and offloading enhancement. To achieve this, a mobile cloud offloading and scheduling problem (MCOSP) is formulated using the mixed linear programming, and offline optimal results are obtained from optimization solving algorithms. Due to the NP-hardness of the proposed problem, an online, lightweight scheduling algorithm based on ski-rental framework was proposed to solve MCOSP at runtime to provide timely offloading and scheduling decisions for the HMC offloading service. Experiments showed that the proposed online algorithm can achieve a 2-competitive performance of the optimal solution.

Chapter 5 and 6 focused on the challenge of maintaining system reliability and increasing user participation to support a seamless and reliable task offloading service in the HMC environment.

In particular, Chapter 5 introduced a group-based fault tolerant mechanism for the proposed HMC system. Due to the heterogeneity of the devices and machines participating, a onefold fault tolerant strategy may downgrade the performance. Therefore, this chapter first proposed a machine grouping algorithm based on the decision tree method to dynamically classify devices and machines into different ability group based on criteria such as capability, reliability, and availability. Then different fault tolerant policies such as replication and checkpointing are applied to the offloaded tasks based on the execution requirement and machine group the tasks are being offloaded. The proposed mechanism is designed as a standalone module that can be added to work with the existing mobile cloud offloading frameworks. Hence, it can take the offloading schedules devised from the existing framework as inputs and generates the fault tolerant policies. The experiments conducted showed that the proposed mechanism outperformed the tradition onefold fault tolerant strategies that adopted on the existing distributed computing systems.

Chapter 6 focused on the incentive issue of the mobile cloud offloading services in the HMC environment. Since mobile users may lack motivations to share their device to run foreign tasks due to the battery concern, a reverse-auction based incentive mechanism was proposed to increase the participation of mobile users and ensure all the partici-

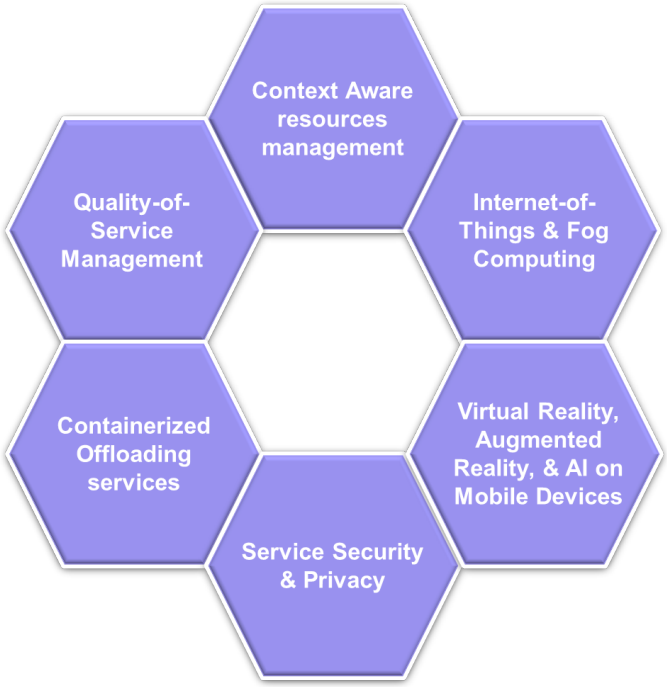


Figure 7.1: Future research directions

pants obtain benefits by offering their redundant computing resources. The proposed algorithms are proved to guarantee computation efficient, truthful, and individual rationality for each participant through mathematical analysis and experiments. A prototype is also implemented on mCloud framework to show its feasibility in practice.

7.2 Future Directions

Nowadays, innovations on mobile devices have gone beyond mobile phones to all types of devices such as wearable devices, smart home appliances, and smart sensors. The connectivity and processing power of these devices make it possible to build a machine-to-machine network where the research works proposed in this thesis can be further extended to. This section gives some insights into a few unexplored research fields. Figure 7.1 summarise the future research directions we propose.

7.2.1 Context Aware Management of Resources

Within the heterogeneous mobile cloud environment, the context information such as user mobility, social information among users, network conditions, and device information can provide additional assistance for decision modules to devise more comprehensive mobile augmentation solutions based on different objectives. The multi-tier HMC framework makes it possible to enable service providers to process the context changes so that the quality of their services can be improved, and bring the opportunities for mobile cloud social-aware applications. However, the rapidly changing execution environment hinders the efficiency of continuous context monitoring and analysis that produce considerable overhead on mobile devices. Therefore, designing resource and energy efficient task allocation and scheduling mechanisms for multi-tier mobile augmentation systems is necessary.

7.2.2 Quality of Service Management

The Quality of Service (QoS) in HMC offloading services refers to the criteria such as service response time (delay), constant wireless communication, availability and scalability of services, the fair use of services, and mobility management. For mobile device based systems, the fundamental problem for improving QoS is mobility management. Hence, an efficient mobility management scheme to estimate mobile device's available time within the system is in crucial need. With the support of mobility management, we can further improve the service response time by utilizing computing resources such as other mobile devices and Cloudlets in vicinity based on the mobile task requirements. On the other hand, it is challenging to design a service model that can manage a large number of mobile clients and wireless communication system while ensuring the availability of services. Therefore, efficient and continuous provisioning of resources and services in mobile augmentation service systems is a research perspective that needs investigation.

7.2.3 Security and Privacy

Due to the data transmission between mobile devices and other computing resources like cloud and mobile devices in the vicinity, data safety, and privacy are important concerns.

Despite an enormous amount of research has been proposed for security issues in the cloud, it is still one of the major gaps in mobile cloud-based systems. First, the security and privacy mechanism needs to be lightweight to reduce the overhead on mobile devices. Second, due to a large amount of data transmission over wireless networks, data protection for integrity and confidentiality in wireless networks need to be considered. Last but not least, a trustworthy distributed computing model is expected to cope with the computation taken place on the remote server and mobile devices and prevent unauthorized access as well as potential data leakage.

7.2.4 Internet-of-Things and Fog Computing

As the wireless technologies and innovations on mobile devices continue to develop, things in our life from refrigerators to cars, or from watches to textbooks, have been equipped with wireless connectivity and a certain level of processing capability. Therefore, the concept of Internet-of-Things (IoT) has come to reality. IoT applications utilize the hierarchical architecture of sensors, smart devices, and cloud services to provide intelligence virtually to everything. An extension to our currently proposed mCloud system framework can be developed to make it compatible with the IoT environment. Mechanisms such as energy efficient resource management in the IoT system and service orchestration on the different layer of the hierarchy can be developed based on the extended framework. Since currently there is no technical standard for IoT implementation, many available technologies such as ZigBee, WiFi, and Bluetooth can be used. Therefore, issues like compatibility with different technologies and multiple cloud vendors can also be further investigated.

7.2.5 Container-based Services

The current mobile cloud computing service models mostly leverage virtual machines (VMs) based cloud resources. One of the issues is that the service usage and VM utilization for the mobile cloud offloading service as the cloud VM instance is usually dedicated to an offloading service user and mobile applications only offload part of the tasks from time to time. Recently, container technology has emerged and gained popularity among

industries. There are several benefits of adopting container technology. Instead of hardware level virtualization, containers use operating system level virtualization. Only the application, and the libraries and file system needed are packed in a container. It enables not only lightweight deployment and easy migration since the size of a container is usually small, but also can increase the scalability and the cross-platform compatibility which will be beneficial to the proposed HMC systems. In order to provide an efficient container powered HMC services, algorithms regarding container service orchestration, and scheduling and migration algorithms need to be investigated.

7.2.6 Virtual Reality, Augmented Reality and Artificial Intelligence on Mobile Devices

The larger display and more powerful hardware on mobile devices make it widely popular to build augmented reality (AR) and virtual reality (VR) mobile applications. These types of mobile applications require constant camera data streaming of the captured frames on cameras and always-on display. In addition, artificial intelligence (AI) technologies are used to enable devices to make cognitive recognition like human beings to discover useful information from the captured data. Recent research has seen the development in this field. FlashBack [19] is a system proposed for VR head-mounted devices to pre-compute and cache all possible encountering images to reduce the computational burden on the device GPU. Furion [126] is a mobile VR framework that enables QoS focused application developing on mobile devices using cloud offloading. MobileDeepPill [249] is a mobile AR system for smartphones to help identify unknown prescription pills captured by the phone's camera by using a proposed deep learning image recognition algorithm. The constant data streaming and high computational requirement of AI from these types of applications yield challenges for mobile devices regarding energy efficiency, data streaming management, pre-stored data management for reuse, and network throughput optimization. Designing efficient mobile cloud offloading system for these applications can provide possible solutions.

7.3 Final Remarks

Cloud computing and innovations on mobile devices have made it possible for mobile users to collaborate using a heterogeneous mobile cloud offloading system. In this thesis, approaches including context-aware task offloading technique, task scheduling algorithms for optimal application performance, adaptive fault tolerant generating algorithm, and auction-based incentive mechanism are investigated and developed to provide an efficient and seamless mobile cloud offloading service. These proposed solutions provide opportunities for further innovation and development in the world of Internet-of-Things and edge computing.

Bibliography

[1] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 337–368, 2014.

[2] S. Abolfazli, Z. Sanaei, M. Shiraz, and A. Gani, "Momcc: Market-oriented architecture for mobile cloud computing based on service oriented architecture," in *Proceedings of 2012 1st IEEE International Conference on Communications in China Workshops (ICCC)*, Aug 2012, pp. 8–13.

[3] R. Achary, V. Vityanathan, P. Raj, and S. Nagarajan, *Dynamic job scheduling using ant colony optimization for mobile cloud computing*. Cham: Springer International Publishing, 2015, pp. 71–82.

[4] E. Ahmed, A. Gani, M. K. Khan, R. Buyya, and S. U. Khan, "Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges," *Journal of Network and Computer Applications*, vol. 52, pp. 154 – 172, 2015.

[5] E. Ahmed, A. Gani, M. Sookhak, S. H. A. Hamid, and F. Xia, "Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges," *Journal of Network and Computer Applications*, vol. 52, pp. 52 – 68, 2015.

[6] F. A. Ali, P. Simoens, T. Verbelen, P. Demeester, and B. Dhoedt, "Mobile device power models for energy efficient dynamic offloading at runtime," *Journal of Systems and Software*, vol. 113, pp. 173 – 187, 2016.

[7] M. Ali, J. M. Zain, M. F. Zolkipli, and G. Badshah, "Mobile cloud computing & mobile battery augmentation techniques: A survey," in *Proceedings of 2014 IEEE Student Conference on Research and Development*, Dec 2014, pp. 1–6.

[8] A. D. Alisa Land, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

[9] O. Alliance, "Osgi the dynamic module system for java," *accessed, May*, vol. 25, 2009.

[10] ——, "Osgi alliance," *Available online http://www.osgi.org/ (accessed 31 November 2010)*, 2010.

[11] H. M. Ammari, "Using group mobility and multihomed mobile gateways to connect mobile ad hoc networks to the global ip internet," *International Journal of Communication Systems*, vol. 19, no. 10, pp. 1137–1165, 2006.

[12] P. Angin and B. K. Bhargava, "An agent-based optimization framework for mobile-cloud computing," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 4, no. 2, pp. 1–17, 2013.

[13] A.-F. Antonescu, A. Gomes, P. Robinson, and T. Braun, "Sla-driven predictive orchestration for distributed cloud-based mobile services," in *Proceedings of 2013 IEEE International Conference on Communications Workshops (ICC)*, June 2013, pp. 738–743.

[14] K. Arnold, J. Gosling, D. Holmes, and D. Holmes, *The Java programming language*. Addison-wesley Reading, 1996, vol. 2.

[15] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 280–293.

[16] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography and application to virus protection," in *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, ser. STOC '95. New York, NY, USA: ACM, 1995, pp. 45–56.

[17] E. Benkhelifa, T. Welsh, L. Tawalbeh, A. Khreishah, Y. Jararweh, and M. Al-Ayyoub, "Ga-based resource augmentation negotation for energy-optimised mobile ad-hoc cloud," in *Proceedings of 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, March 2016, pp. 110–116.

[18] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of 2007 IEEE Symposium on Security and Privacy (SP '07)*, May 2007, pp. 321–334.

[19] K. Boos, D. Chu, and E. Cuervo, "Flashback: Immersive virtual reality on mobile devices via rendering memoization," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. New York, NY, USA: ACM, 2016, pp. 291–304.

[20] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. New York, NY, USA: Cambridge University Press, 1998.

[21] A. Bourd. (2016, March) The opencl specification. Khronos OpenCL Working Group. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-2.2.pdf

[22] D. Box and T. Pattison, *Essential .NET: The Common Language Runtime*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[23] D. C. Brabham, *Crowdsourcing*. Wiley Online Library.

[24] T. D. Braun, H. J. Siegel *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[25] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[26] H. Byun, B.-K. Park, and Y.-S. Jeong, *Mobile agent oriented service for offloading on mobile cloud computing*. Singapore: Springer Singapore, 2017, pp. 920–925.

[27] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad-hoc network research," *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002.

[28] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, 2010.

[29] V. Cardellini, V. De Nitto Personé, V. Di Valerio, F. Facchinei, V. Grassi, F. Lo Presti, and V. Piccialli, "A game-theoretic approach to computation offloading in mobile cloud computing," *Mathematical Programming*, vol. 157, no. 2, pp. 421–449, 2016.

[30] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transsaction on Computer Systems*, vol. 3, no. 1, pp. 63–75, Feb. 1985.

[31] C. Chang, S. N. Srirama, and S. Ling, *An Adaptive Mediation Framework for Mobile P2P Social Content Sharing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 374–388.

[32] M. Chase and S. S. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2009, pp. 121–130.

[33] C. Chen, W. Bao, X. Zhu, H. Ji, W. Xiao, and J. Wu, "Agile: A terminal energy efficient scheduling method in mobile cloud computing," *Transactions on Emerging Telecommunications Technologies*, vol. 26, no. 12, pp. 1323–1336, 2015.

[34] C.-A. Chen, M. Won, R. Stoleru, and G. G. Xie, "Energy-efficient fault-tolerant data storage and processing in mobile cloud," *IEEE Transactions on Cloud Computing*, vol. 3, no. 1, pp. 28–41, Jan 2015.

[35] S. Chen, Y. Wang, and M. Pedram, "A semi-markovian decision process based control method for offloading tasks from mobile devices to the cloud," in *Proceedings of 2013 IEEE Global Communications Conference (GLOBECOM)*, Dec 2013, pp. 2885–2890.

[36] X. Chen, I. Beschastnikh, L. Zhuang, F. Yang, Z. Qian, L. Zhou, G. Shen, and J. Shen, "Sonora: A platform for continuous mobile-cloud computing," Tech. Rep., March 2012.

[37] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, April 2015.

[38] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[39] Z. Cheng, P. Li, J. Wang, and S. Guo, "Just-in-time code offloading for wearable computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 1, pp. 74–83, March 2015.

[40] D. Chess, C. Harrison, and A. Kershenbaum, *Mobile agents: Are they a good idea?* Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 25–45.

[41] S. Choi, K. Chung, and H. Yu, "Fault tolerance and qos scheduling using can in mobile social cloud computing," *Cluster Computing*, vol. 17, no. 3, pp. 911–926, 2014.

[42] S. Choi, M. Baik, J. Gil, S. Jung, and C. Hwang, "Adaptive group scheduling mechanism using mobile agents in peer-to-peer grid computing environment," *Applied Intelligence*, vol. 25, no. 2, pp. 199–221, 2006.

[43] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the 6th conference on Computer systems*. ACM, 2011, pp. 301–314.

[44] E. Cinlar, *Introduction to stochastic processes*. Courier Corporation, 2013.

[45] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286.

[46] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot, "Optimized link state routing protocol (olsr)," 2003.

[47] P. Clements, T. Papaioannou, and J. Edwards, "Aglets: Enabling the virtual enterprise," in *Proceedings of the Managing Enterprises-Stakeholders, Engineering, Logistics and Achievement InternationalConference (MESELA97)*, 1997.

[48] D. W. Coit and J. C. Liu, "System realiability optimization with k-out-of-n subsystems," *International Journal of Reliability, Quality and Safety Engineering*, vol. 07, no. 02, pp. 129–142, 2000.

[49] comScore, "The 2017 u.s. cross-platform future in focus," Tech. Rep., 2017. [Online]. Available: http://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/2017-US-Cross-Platform-Future-in-Focus

[50] ——, "Mobiles hierarchy of needs," Tech. Rep., 2017. [Online]. Available: http://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/Mobiles-Hierarchy-of-Needs-MMA-Forum-Singapore-2017

[51] M. Conti and S. Giordano, "Mobile ad hoc networking: Milestones, challenges, and new research directions," *IEEE Communications Magazine*, vol. 52, no. 1, pp. 85–96, January 2014.

[52] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[53] Y. Cui, Z. Lai, X. Wang, N. Dai, and C. Miao, "Quicksync:improving synchronization efficiency for mobile cloud storage services," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. New York, NY, USA: ACM, 2015, pp. 592–603.

[54] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *ACM Communication*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[55] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3317–3329, Dec 2015.

[56] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[57] M. Dobber, R. van der Mei, and G. Koole, "Dynamic load balancing and job replication in a global-scale grid environment: A comparison," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 2, pp. 207–218, Feb 2009.

[58] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, and V. H. Tuulos, "Misco: A mapreduce framework for mobile systems," in *Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments*, ser. PETRA '10. New York, NY, USA: ACM, 2010, pp. 32:1–32:8.

[59] C. Doukas, T. Pliakas, and I. Maglogiannis, "Mobile healthcare information management utilizing cloud computing and android os," in *Proceedings of 2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, Aug 2010, pp. 1037–1040.

[60] R. O. Duda, P. E. Hart, D. G. Stork *et al.*, *Pattern classification*. Wiley New York, 1973, vol. 2.

[61] N. Eagle, A. S. Pentland, and D. Lazer, "Inferring friendship network structure by using mobile phone data," *Proceedings of the National Academy of Sciences*, vol. 106, no. 36, pp. 15 274–15 278, 2009.

[62] N. Eagle and A. (Sandy) Pentland, "Reality mining: sensing complex social systems," *Personal and Ubiquitous Computing*, vol. 10, no. 4, pp. 255–268, May 2006.

[63] D. Ehringer, "The dalvik virtual machine architecture," *Technical report*, vol. 4, p. 8, 2010.

[64] H. Eom, P. S. Juste, R. Figueiredo, O. Tickoo, R. Illikkal, and R. Iyer, "Opencl-based remote offloading framework for trusted mobile cloud computing," in *Proceedings of 2013 International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2013, pp. 240–248.

[65] X. Fan, C. S. Ellis, and A. R. Lebeck, *The Synergy Between Power-Aware Memory Systems and Processor Voltage Scaling*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 164–179.

[66] D. Fernández-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, 1989.

[67] N. Fernando, S. W. Loke, and W. Rahayu, "Dynamic mobile cloud computing: ad hoc and opportunistic job sharing," in *Proceedings of 2011 4th IEEE International Conference on Utility and Cloud Computing (UCC)*, Dec 2011, pp. 281–286.

[68] ——, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[69] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of cloudlets on interactive mobile cloud applications," in *Proceedings of the 16th IEEE International Enterprise Distributed Object Computing Conference*. IEEE, 2012, pp. 123–132.

[70] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proceedings of the 22nd International Conference onDistributed Computing Systems*, 2002, pp. 217–226.

[71] H. Flores and S. N. Srirama, "Mobile cloud middleware," *Journal of Systems and Software*, vol. 92, pp. 82 – 94, 2014.

[72] K. Gai, M. Qiu, and H. Zhao, "Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing," *IEEE Transactions on Cloud Computing*, 2017.

[73] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong, "Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing," *Journal of Network and Computer Applications*, vol. 59, pp. 46 – 54, 2016.

[74] X. Gan, Y. Li, W. Wang, L. Fu, and X. Wang, "Social crowdsourcing to friends: An incentive mechanism for multi-resource sharing," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 3, pp. 795–808, March 2017.

[75] A. Ganesh, M. Sandhya, and S. Shankar, "A study on fault tolerance methods in cloud computing," in *Proceedings of 2014 IEEE International Advance Computing Conference (IACC)*, Feb 2014, pp. 844–849.

[76] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[77] P. Garg and V. Sharma, "An efficient and secure data storage in mobile cloud computing through rsa and hash function," in *Proceedings of 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, Feb 2014, pp. 334–339.

[78] A. Goldsmith, *Wireless communications*. Cambridge university press, 2005.

[79] M. S. Gordon, D. K. Hong, P. M. Chen, J. Flinn, S. Mahlke, and Z. M. Mao, "Accelerating mobile applications through flip-flop replication," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '15. New York, NY, USA: ACM, 2015, pp. 137–150.

[80] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "COMET: Code offload by migrating execution transparently," in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*. Hollywood, CA: USENIX, 2012.

[81] M. Goudarzi, M. Zamani, and A. T. Haghighat, "A fast hybrid multi-site computation offloading for mobile cloud computing," *Journal of Network and Computer Applications*, vol. 80, pp. 219 – 231, 2017.

[82] M. Goudarzi, M. Zamani, and A. Toroghi Haghighat, "A genetic-based decision algorithm for multisite computation offloading in mobile cloud computing," *International Journal of Communication Systems*, 2016.

[83] M. Goyal and P. Saini, "A fault-tolerant energy-efficient computational offloading approach with minimal energy and response time in mobile cloud computing," in *Proceedings of 2016 4th International Conference on Parallel, Distributed and Grid Computing (PDGC)*, Dec 2016, pp. 44–49.

[84] M. Green and G. Ateniese, *Identity-Based Proxy Re-encryption*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 288–306.

[85] W. Gropp, E. Lusk, and A. Skjellum, "Using mpi: portable parallel programming with the message-passing interface," 1994.

[86] G. Guerrero-Contreras, J. L. Garrido, S. Balderas-Diaz, and C. Rodriguez-Dominguez, "A context-aware architecture supporting service availability in mobile cloud computing," *IEEE Transactions on Services Computing*, 2017.

[87] Q. B. Hani and J. P. Dichter, "Energy-efficient service-oriented architecture for mobile cloud handover," *Journal of Cloud Computing*, vol. 6, no. 1, p. 9, 2017.

[88] D. T. Hoang, D. Niyato, and P. Wang, "Optimal admission control policy for mobile cloud computing hotspot with cloudlet," in *Proceedings of 2012 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2012, pp. 3145–3149.

[89] B. Hong and V. Prasanna, "Adaptive allocation of independent tasks to maximize throughput," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1420–1435, Oct 2007.

[90] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Mobile Cloud Computing*, ser. MCC '13. New York, NY, USA: ACM, 2013, pp. 15–20.

[91] X. Hong, M. Gerla, G. Pei, and C.-C. Chiang, "A group mobility model for ad hoc wireless networks," in *Proceedings of the 2nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM '99. New York, NY, USA: ACM, 1999, pp. 53–60.

[92] C.-W. Huang, M. Chen, and D. Zavin. (2015) Android-x86 - porting android to x86. [Online]. Available: http://www.android-x86.org/

[93] D. Huang, Z. Zhou, L. Xu, T. Xing, and Y. Zhong, "Secure data processing framework for mobile cloud computing," in *Proceedings of 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2011, pp. 614–618.

[94] D. Huang, X. Zhang, M. Kang, and J. Luo, "Mobicloud: Building secure cloud framework for mobile computing and communication," in *Proceedings of 2010 5th*

IEEE International Symposium on Service Oriented System Engineering (SOSE), June 2010, pp. 27–34.

[95] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, ser. MCS '10. New York, NY, USA: ACM, 2010, pp. 6:1–6:5.

[96] K. A. Hummel and G. Jelleschitz, "A robust decentralized job scheduling approach for mobile peers in ad-hoc grids," in *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, May 2007, pp. 461–470.

[97] S.-H. Hung, T.-T. Tzeng, G.-D. Wu, and J.-P. Shieh, "A code offloading scheme for big-data processing in android applications," *Software: Practice and Experience*, vol. 45, no. 8, pp. 1087–1101, Aug. 2015.

[98] C.-L. Hwang, Y.-J. Lai, and T.-Y. Liu, "A new approach for multiple objective decision making," *Computers Operations Research*, vol. 20, no. 8, pp. 889 – 899, 1993.

[99] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM (JACM)*, vol. 24, no. 2, pp. 280–289, 1977.

[100] IBM. (2016) Ibm netezza data warehouse appliances. [Online]. Available: https://www-01.ibm.com/software/data/netezza/

[101] W. Itani, A. Kayssi, and A. Chehab, "Energy-efficient incremental integrity for securing storage in mobile cloud computing," in *Proceedings of 2010 International Conference on Energy Aware Computing*, Dec 2010, pp. 1–2.

[102] J. James E. Kelley, "Critical-path planning and scheduling: Mathematical basis," *Operations Research*, vol. 9, no. 3, pp. 296–320, 1961.

[103] J. Jeong, H. Jeong, E. Lee, T. Oh, and D. Du, "Saint: Self-adaptive interactive navigation tool for cloud-based vehicular traffic optimization," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 4053–4067, June 2016.

[104] W. Jia, H. Zhu, Z. Cao, L. Wei, and X. Lin, "Sdsm: A secure data service mechanism in mobile cloud computing," in *Proceedings of 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2011, pp. 1060–1065.

[105] A. L. Jin, W. Song, P. Wang, D. Niyato, and P. Ju, "Auction mechanisms toward efficient resource sharing for cloudlets in mobile cloud computing," *IEEE Transactions on Services Computing*, vol. 9, no. 6, pp. 895–909, Nov 2016.

[106] J. Jubin and J. D. Tornow, "The darpa packet radio network protocols," *Proceedings of the IEEE*, vol. 75, no. 1, pp. 21–32, Jan 1987.

[107] W. Junior, A. Frana, K. Dias, and J. N. de Souza, "Supporting mobility-aware computational offloading in mobile cloud environment," *Journal of Network and Computer Applications*, vol. 94, pp. 93 – 108, 2017.

[108] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, 2017.

[109] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive randomized algorithms for non-uniform problems," in *Proceedings of the 1st annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1990, pp. 301–309.

[110] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, "Competitive snoopy caching," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, Oct 1986, pp. 244–254.

[111] E. Kartal Tabak, B. Barla Cambazoglu, and C. Aykanat, "Improving the performance of independent task assignment heuristics minmin,maxmin and sufferage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1244–1256, May 2014.

[112] M. Kaya, A. Koyiit, and P. E. Eren, "An adaptive mobile cloud computing framework using a call graph based model," *Journal of Network and Computer Applications*, vol. 65, pp. 12 – 35, 2016.

[113] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, *Cuckoo: A computation offloading framework for smartphones*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 59–79.

[114] A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 393–413, 2014.

[115] A. N. Khan, M. L. M. Kiah, S. A. Madani, M. Ali, A. u. R. Khan, and S. Shamshirband, "Incremental proxy re-encryption scheme for mobile cloud computing environment," *The Journal of Supercomputing*, vol. 68, no. 2, pp. 624–651, 2014.

[116] A. N. Khan, M. M. Kiah, S. U. Khan, and S. A. Madani, "Towards secure mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1278–1299, 2013.

[117] A. N. Khan, M. L. Mat Kiah, S. A. Madani, A. u. R. Khan, and M. Ali, "Enhanced dynamic credential generation scheme for protection of user identity in mobile-cloud computing," *The Journal of Supercomputing*, vol. 66, no. 3, pp. 1687–1706, 2013.

[118] T. Kirkham, S. Ravet, S. Winfield, and S. Kellomki, "A personal data store for an internet of subjects," in *Proceedings of 2011 International Conference on Information Society (i-Society)*, June 2011, pp. 92–97.

[119] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Transactions on Computer Systems*, vol. 10, no. 1, pp. 3–25, Feb. 1992.

[120] A. Klein, C. Mannweiler, J. Schneider, and H. D. Schotten, "Access schemes for mobile cloud computing," in *Proceedings of 2010 11th International Conference on Mobile Data Management*, May 2010, pp. 387–392.

[121] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 1, pp. 23–31, Jan 1987.

[122] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,"

in *Proceedings of 2012 IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2012, pp. 945–953.

[123] D. Kotz, T. Henderson, I. Abyzov, and J. Yeo, "CRAWDAD dataset dartmouth/campus (v.2009-09-09)," Downloaded from http://crawdad.org/dartmouth/campus/20090909/syslog, Sep. 2009, traceset: syslog.

[124] M. D. Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *Proceedings of 2010 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, March 2010, pp. 217–226.

[125] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, April 2010.

[126] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," in *Proceedings of the 23rd International Conference on Mobile Computing and Networking*. Snowbird, Utah, USA: ACM, October 2017.

[127] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[128] J. F. Lawless, *Statistical models and methods for lifetime data*. John Wiley & Sons, 2011, vol. 362.

[129] J. Lee, S. Choi, T. Suh, H. Yu, and J. Gil, "Group-based scheduling algorithm for fault tolerance in mobile grid," in *Security-Enriched Urban Computing and Smart Grid*. Springer, 2010, pp. 394–403.

[130] J. Lee and D. Hong, "Pervasive forensic analysis based on mobile cloud computing," in *Proceedings of 2011 Third International Conference on Multimedia Information Networking and Security*, Nov 2011, pp. 572–576.

[131] G. Lewis and P. Lago, "A catalog of architectural tactics for cyber-foraging," in *Proceedings of 2015 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*, May 2015, pp. 53–62.

[132] B. Li, Y. Pei, H. Wu, and B. Shen, "Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds," *The Journal of Supercomputing*, vol. 71, no. 8, pp. 3009–3036, Aug 2015.

[133] ——, "Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds," *The Journal of Supercomputing*, vol. 71, no. 8, pp. 3009–3036, 2015.

[134] C. Li and L. Li, "Phased scheduling for resource-constrained mobile devices in mobile cloud computing," *Wireless Personal Communications*, vol. 77, no. 4, pp. 2817–2837, 2014.

[135] F. Li, Y. Rahulamathavan, M. Rajarajan, and R. C.-W. Phan, "Low complexity multi-authority attribute based encryption scheme for mobile cloud computing," in *Proceedings of 2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, March 2013, pp. 573–577.

[136] J. Li, K. Bu, X. Liu, and B. Xiao, "Enda: Embracing network inconsistency for dynamic application offloading in mobile cloud computing," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Mobile Cloud Computing*. New York, NY, USA: ACM, 2013, pp. 39–44.

[137] W. Li, Y. Zhao, S. Lu, and D. Chen, "Mechanisms and challenges on mobility-augmented service provisioning for mobile cloud computing," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 89–97, March 2015.

[138] X. Li, H. Zhang, and Y. Zhang, "Deploying mobile computation in cloud service," in *Proceedings of the 1st International Conference on Cloud Computing*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 301–311.

[139] Y. Li, M. Chen, W. Dai, and M. Qiu, "Energy optimization with dynamic task scheduling mobile cloud computing," *IEEE Systems Journal*, vol. PP, no. 99, pp. 1–10, 2015.

[140] H. Liang, D. Huang, and D. Peng, *On Economic Mobile Cloud Computing Model*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 329–341.

[141] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 175–186, March 2015.

[142] A. Litke, D. Skoutas, K. Tserpes, and T. Varvarigou, "Efficient task replication and management for adaptive fault tolerance in mobile grid environments," *Future Generation Computer Systems*, vol. 23, no. 2, pp. 163 – 178, 2007.

[143] K. Liu, J. Peng, H. Li, X. Zhang, and W. Liu, "Multi-device task offloading with time-constraints for energy efficiency in mobile cloud computing," *Future Generation Computer Systems*, pp. 1 – 14, 2016.

[144] Q. Liu, X. Jian, J. Hu, H. Zhao, and S. Zhang, "An optimized solution for mobile environment using mobile cloud computing," in *Proceedings of 2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*, Sept 2009, pp. 1–5.

[145] X. Liu, C. Yuan, Z. Yang, and Z. Zhang, "Mobile-agent-based energy-efficient scheduling with dynamic channel acquisition in mobile cloud computing," *Journal of Systems Engineering and Electronics*, vol. 27, no. 3, pp. 712–720, June 2016.

[146] Y. Liu, M. J. Lee, and Y. Zheng, "Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system," *IEEE Transactions on Mobile Computing*, vol. 15, no. 10, pp. 2398–2410, Oct 2016.

[147] Y. Liu, C. Xu, Y. Zhan, Z. Liu, J. Guan, and H. Zhang, "Incentive mechanism for computation offloading using edge computing: A stackelberg game approach," *Computer Networks*, 2017.

[148] R. K. Lomotey and R. Deters, "Reliable consumption of web services in a mobile-cloud ecosystem using rest," in *Proceedings of 2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, March 2013, pp. 13–24.

[149] Y. Lu, B. Zhou, L.-C. Tung, M. Gerla, A. Ramesh, and L. Nagaraja, "Energy-efficient content retrieval in mobile cloud," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Mobile Cloud Computing*. New York, NY, USA: ACM, 2013, pp. 21–26.

[150] C. M. MacKenzie and K. Laskey, "Reference model for service oriented architecture 1.0," 2006.

[151] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA., 1967, pp. 281–297.

[152] A. Malhotra, S. K. Dhurandher, and B. Kumar, "Resource allocation in multi-hop mobile ad hoc cloud," in *Proceedings of 2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, March 2014, pp. 1–6.

[153] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, Dec 2016.

[154] E. E. Marinelli, "Hyrax: cloud computing on mobile devices using mapreduce," Ph.D. dissertation, Carnegie Mellon University, 2009.

[155] C. X. Mavromoustakis *et al.*, "A social-oriented mobile cloud scheme for optimal energy conservation," *Resource Management of Mobile Cloud Computing Networks and Environments*, pp. 97–121, 2015.

[156] A. Mayberry, Y. Tun, P. Hu, D. Smith-Freedman, D. Ganesan, B. M. Marlin, and C. Salthouse, "Cider: Enabling robustness-power tradeoffs on a computational eyeglass," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. New York, NY, USA: ACM, 2015, pp. 400–412.

[157] G. McCluskey, "Using java reflection," *Java Developer Connection*, 1998.

[158] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 346–356.

[159] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[160] H. Miyake and N. Kami, "Qoi-based data upload control for mobility-aware cloud services," in *Proceedings of 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, March 2015, pp. 16–23.

[161] J. P. Morrison, *Flow-based programming: a new approach to application development*. CreateSpace, 2010.

[162] R. B. Myerson and M. A. Satterthwaite, "Efficient mechanisms for bilateral trading," *Journal of Economic Theory*, vol. 29, no. 2, pp. 265 – 281, 1983.

[163] N. Naksinehaboon, Y. Liu, C. Leangsuksun, R. Nassar, M. Paun, and S. L. Scott, "Reliability-aware approach: An incremental checkpoint/restart model in hpc environments," in *Proceedings of 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, May 2008, pp. 783–788.

[164] Y. Nan, W. Li, W. Bao, F. C. Delicato, P. F. Pires, and A. Y. Zomaya, "Cost-effective processing for delay-sensitive applications in cloud of things systems," in *Proceedings of 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, Oct 2016, pp. 162–169.

[165] Nectar, *URL: https://nectar.org.au/research-cloud/*, 2015.

[166] B. J. Nelson, "Remote procedure call," Ph.D. dissertation, 1981.

[167] J. M. Ng and Y. Zhang, "A mobility model with group partitioning for wireless ad hoc networks," in *Proceedings of the 3 rd International Conference on Information Technology and Applications (ICITA'05)*, vol. 2, July 2005, pp. 289–294.

[168] G. Optimization *et al.*, "Gurobi optimizer reference manual," *URL: http://www.gurobi. com*, 2015.

[169] Oracle, *Java Remote Method Invocation*, may 2016. [Online]. Available: http://docs.oracle.com/javase/6/docs/technotes/guides/rmi/index.html

[170] R. K. Panta, R. Jana, F. Cheng, Y.-F. R. Chen, and V. A. Vaishampayan, "Phoenix: Storage using an autonomous mobile infrastructure," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1863–1873, Sept 2013.

[171] J. Park, H. Yu, K. Chung, and E. Lee, "Markov chain based monitoring service for fault tolerance in mobile cloud computing," in *Proceedings of 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA)*, March 2011, pp. 520–525.

[172] J. Park, H. Yu, H. Kim, and E. Lee, "Dynamic group-based fault tolerance technique for reliable resource management in mobile cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 10, pp. 2756–2769, 2016.

[173] S. Parsons, J. A. Rodriguez-Aguilar, and M. Klein, "Auctions and bidding: A guide for computer scientists," *ACM Computing Surveys*, vol. 43, no. 2, pp. 10:1–10:59, Feb. 2011.

[174] A. Pashtan, *Mobile web services*. Cambridge University Press, 2005.

[175] J. Pearl, *Heuristics: Intelligent search strategies for computer problem solving*. Addison-Wesley Pub. Co., Inc.,Reading, MA, Jan 1984.

[176] K. Pearson, "The problem of the random walk," *Nature*, vol. 72, no. 1865, p. 294, 1905.

[177] G. Pei, M. Gerla, and T.-W. Chen, "Fisheye state routing: A routing scheme for ad hoc wireless networks," in *Proceedings of 2000 IEEE International Conference on Communications*, vol. 1. IEEE, 2000, pp. 70–74.

[178] T. Penner, A. Johnson, B. V. Slyke, M. Guirguis, and Q. Gu, "Transient clouds: Assignment and collaborative execution of tasks on mobile devices," in *Proceedings of 2014 IEEE Global Communications Conference*, Dec 2014, pp. 2801–2806.

[179] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," Tech. Rep., 2003.

[180] Q. Qi, J. Liao, J. Wang, Q. Li, and Y. Cao, "Dynamic resource orchestration for multi-task application in heterogeneous mobile cloud computing," in *Proceedings of 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2016, pp. 221–226.

[181] Y. Qiao, Y. Cheng, J. Yang, J. Liu, and N. Kato, "A mobility analytical framework for big mobile data in densely populated area," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 2, pp. 1443–1455, Feb 2017.

[182] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[183] M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. 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*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 83–90.

[184] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "MuSIC: Mobility-aware optimal service allocation in mobile cloud computing," in *Proceedings of the 6th IEEE International Conference on Cloud Computing*. IEEE, 2013, pp. 75–82.

[185] S. Rallapalli, A. Ganesan, K. Chintalapudi, V. N. Padmanabhan, and L. Qiu, "Enabling physical analytics in retail stores using smart glasses," in *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*. New York, NY, USA: ACM, 2014, pp. 115–126.

[186] S. Rashidi and S. Sharifian, "A hybrid heuristic queue based algorithm for task assignment in mobile cloud," *Future Generation Computer Systems*, vol. 68, pp. 331 – 345, 2017.

[187] A. Ravi and S. K. Peddoju, "Handoff strategy for improving energy efficiency and cloud service availability for mobile devices," *Wireless Personal Communications*, pp. 1–32, 2014.

[188] H. Reading, "The mobile cloud market and outlook to 2017," Heavy Reading, Tech. Rep., 2013.

[189] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-osgi: Distributed applications through software modularization," in *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2007, pp. 1–20.

[190] J. S. Rellermeyer, O. Riva, and G. Alonso, "Alfredo: An architecture for flexible interaction with electronic devices," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2008, pp. 22–41.

[191] H. Rinne, *The Weibull distribution: a handbook*. CRC Press, 2008.

[192] S. Robinson, "Cellphone energy gap: desperately seeking solutions," *Strategy Analytics*, 2009.

[193] C. Rossi, M. H. Heyi, and F. Scullino, "A service oriented cloud-based architecture for mobile geolocated emergency services," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 11, 2017.

[194] T. L. Saaty, "The analytic hierarchy process: planning, priority setting, resources allocation," *New York: McGraw*, 1980.

[195] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: taxonomy and open challenges," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 369–392, First 2014.

[196] Z. Sanaei, S. Abolfazli, A. Gani, and M. Shiraz, "Sami: Service-based arbitrated multi-tier infrastructure for mobile cloud computing," in *Proceedings of 2012 1st IEEE International Conference on Communications in China Workshops (ICCC)*, Aug 2012, pp. 14–19.

[197] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, June 2015.

[198] M. Satyanarayanan, "Pervasive computing: vision and challenges," *IEEE Personal Communications*, vol. 8, no. 4, pp. 10–17, Aug 2001.

[199] ——, "Mobile computing: the next decade," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 15, no. 2, pp. 2–10, Aug. 2011.

[200] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[201] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: Computation offloading as a service for mobile devices," in *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. New York, NY, USA: ACM, 2014, pp. 287–296.

[202] C.-S. Shih, Y.-H. Wang, and N. Chang, "Multi-tier elastic computation framework for mobile cloud computing," in *Proceedings of 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, March 2015, pp. 223–232.

[203] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1294–1313, Third 2013.

[204] M. Shiraz, A. Gani, A. Shamim, S. Khan, and R. W. Ahmad, "Energy efficient computational offloading framework for mobile cloud computing," *Journal of Grid Computing*, vol. 13, no. 1, pp. 1–18, 2015.

[205] J. Shuja, A. Gani, M. H. Rehman, E. Ahmed, S. A. Madani, M. K. Khan, and K. Ko, "Towards native code offloading based mcc frameworks for multimedia applications: A survey," *Journal of Network and Computer Applications*, vol. 75, pp. 335 – 354, 2016.

[206] F. A. Silva, P. Maciel, R. Matos *et al.*, "A scheduler for mobile cloud based on weighted metrics and dynamic context evaluation," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 569–576.

[207] S. Soo, C. Chang, and S. N. Srirama, "Proactive service discovery in fog computing using mobile ad hoc social network in proximity," in *Proceedings of 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Dec 2016, pp. 561–566.

[208] M. Sookhak, F. R. Yu, M. K. Khan, Y. Xiang, and R. Buyya, "Attribute-based data access control in mobile cloud computing: Taxonomy and open issues," *Future Generation Computer Systems*, vol. 72, pp. 273 – 287, 2017.

[209] T. Soyata, R. Muraleedharan, J. Langdon, C. Funai, S. Ames, M. Kwon, and W. Heinzelman, "Combat: Mobile-cloud-based compute/communications infrastructure for battlefield applications," in *SPIE defense, security, and sensing*. International Society for Optics and Photonics, 2012.

[210] S. N. Srirama, "Mobile web and cloud services enabling internet of things," *CSI Transactions on ICT*, vol. 5, no. 1, pp. 109–117, Mar 2017.

[211] S. N. Srirama, M. Jarke, and W. Prinz, "Mobile web service provisioning," in *Proceedings of Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW'06)*, Feb 2006, pp. 120–120.

[212] R. W. Stevens, *Unix Network Programming*. Prentice Hall PTR, 1990.

[213] I. Stojmenovic, *Handbook of wireless networks and mobile computing*. John Wiley & Sons, 2003, vol. 27.

[214] T. Taleb, A. Ksentini, and P. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Transactions on Cloud Computing*, 2016.

[215] L. Tang, S. He, and Q. Li, "Double-sided bidding mechanism for resource sharing in mobile cloud," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 2, pp. 1798–1809, Feb 2017.

[216] M. B. Terefe, H. Lee, N. Heo, G. C. Fox, and S. Oh, "Energy-efficient multisite offloading policy using markov decision process for mobile cloud computing," *Pervasive and Mobile Computing*, vol. 27, pp. 75 – 89, 2016.

[217] F. Tian, B. Liu, X. Sun, X. Zhang, G. Cao, and G. Lin, "Movement-based incentive for crowdsourcing," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 8, pp. 7223 – 7233, 2017.

[218] C. K. Toh, *Ad hoc mobile wireless networks: protocols and systems*. Pearson Education, 2001.

[219] A. Toninelli, A. Pathak, and V. Issarny, *Yarta: A middleware for managing mobile social ecosystems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 209–220.

[220] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[221] M. Treaster, "A survey of fault-tolerance and fault-recovery techniques in parallel systems," *arXiv.org*, vol. abs/cs/0501002, 2005.

[222] W. Trneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl, and E. Elmroth, "Dynamic application placement in the mobile cloud network," *Future Generation Computer Systems*, vol. 70, pp. 163 – 177, 2017.

[223] M. Velasquez and P. T. Hester, "An analysis of multi-criteria decision making methods," *International Journal of Operations Research*, vol. 10, no. 2, pp. 56–66, 2013.

[224] T. Verbelen, P. Simoens, F. D. Turck, and B. Dhoedt, "Aiolos: Middleware for improving mobile application performance through cyber foraging," *Journal of Systems and Software*, vol. 85, no. 11, pp. 2629 – 2639, 2012.

[225] ——, "Adaptive deployment and configuration for mobile augmented reality in the cloudlet," *Journal of Network and Computer Applications*, vol. 41, pp. 206 – 216, 2014.

[226] B. Wang, B. Li, and H. Li, "Public auditing for shared data with efficient user revocation in the cloud," in *Proceedings of 2013 IEEE International Conference on Computer Communications (INFOCOM)*, April 2013, pp. 2904–2912.

[227] S. Wang, T. Lei, L. Zhang, C.-H. Hsu, and F. Yang, "Offloading mobile data traffic for qos-aware service provision in vehicular cyber-physical systems," *Future Generation Computer Systems*, vol. 61, pp. 118 – 127, 2016.

[228] W. Wang, P. Xu, and L. T. Yang, "One-pass anonymous key distribution in batch for secure real-time mobile services," in *Proceedings of 2015 IEEE International Conference on Mobile Services*, June 2015, pp. 158–165.

[229] X. Wang, W. Xu, and Z. Jin, "A hidden markov model based dynamic scheduling approach for mobile cloud telemonitoring," in *Proceedings of 2017 IEEE EMBS International Conference on Biomedical Health Informatics (BHI)*, Feb 2017, pp. 273–276.

[230] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, Oct 2016.

[231] Z. Wang, R.-C. Hou, and Z.-M. Zhou, "An android/osgi-based mobile gateway for body sensor network," in *Proceedings of 2016 15th International Symposium on Parallel and Distributed Computing (ISPDC)*, July 2016, pp. 135–140.

[232] X. Wei, J. Fan, T. Wang, and Q. Wang, "Efficient application scheduling in mobile cloud computing based on max–min ant system," *Soft Computing*, vol. 20, no. 7, pp. 2611–2625, 2016.

[233] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM)*, Orlando, Florida, USA, 2012.

[234] Wikipedia. (2017, 04) Instructions per df.

[235] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[236] H. Wu, Q. Wang, and K. Wolter, "Methods of cloud-path selection for offloading in mobile cloud computing systems," in *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science*, Dec 2012.

[237] Q. Xia, W. Liang, Z. Xu, and B. Zhou, "Online algorithms for location-aware task offloading in two-tiered mobile cloud environments," in *Proceedings of the 2014*

IEEE/ACM 7th International Conference on Utility and Cloud Computing. Washington, DC, USA: IEEE Computer Society, 2014, pp. 109–116.

[238] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li, "Ready, set, go: Coalesced offloading from mobile devices to the cloud," in *Proceedings of 2014 IEEE Conference on Computer Communications*, April 2014, pp. 2373–2381.

[239] K. Xie, X. Wang, G. Xie, D. Xie, J. Cao, Y. Ji, and J. Wen, "Distributed multi-dimensional pricing for efficient application offloading in mobile cloud computing," *IEEE Transactions on Services Computing*, 2017.

[240] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2866–2880, Oct 2016.

[241] D. Yang, X. Fang, and G. Xue, "Truthful auction for cooperative communications with revenue maximization," in *Proceedings of 2012 IEEE International Conference on Communications (ICC)*, June 2012, pp. 4888–4892.

[242] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2253–2266, Aug 2015.

[243] L. Yang, J. Cao, G. Liang, and X. Han, "Cost-aware service placement and load dispatching in mobile cloud systems," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1440–1452, May 2016.

[244] S. Yang, X. Bei, Y. Zhang, and Y. Ji, "Application offloading based on r-osgi in mobile cloud computing," in *Proceedings of 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, March 2016, pp. 46–52.

[245] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, and S. Guizani, "Mobile ad hoc cloud: A survey," *Wireless Communications and Mobile Computing*, vol. 16, no. 16, pp. 2572–2589, 2016.

[246] K. Yoon, "A reconciliation among discrete compromise solutions," *Journal of the Operational Research Society*, vol. 38, no. 3, pp. 277–286, 1987.

[247] J. W. Young, "A first order approximation to the optimum checkpoint interval," *ACM Communications*, vol. 17, no. 9, pp. 530–531, Sep. 1974.

[248] J. Yuan and S. Yu, "Efficient public integrity checking for cloud data sharing with multi-user modification," in *Proceedings of 2014 IEEE Conference on Computer Communications*, April 2014, pp. 2121–2129.

[249] X. Zeng, K. Cao, and M. Zhang, "Mobiledeeppill: A small-footprint mobile deep learning system for recognizing unconstrained pill images," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '17. New York, NY, USA: ACM, 2017, pp. 56–67.

[250] L. Zhang, B. Tiwana, R. Dick, Z. Qian, Z. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Oct 2010, pp. 105–114.

[251] W. Zhang, S. Tan, F. Xia, X. Chen, Z. Li, Q. Lu, and S. Yang, "A survey on decision making for task migration in mobile cloud environments," *Personal Ubiquitous Computing*, vol. 20, no. 3, pp. 295–309, Jun. 2016.

[252] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Networks and Applications*, vol. 16, no. 3, pp. 270–284, 2011.

[253] W. Zheng, P. Xu, X. Huang, and N. Wu, "Design a cloud storage platform for pervasive computing environments," *Cluster Computing*, vol. 13, no. 2, pp. 141–151, 2010.

[254] L. Zhou, Z. Yang, J. J. P. C. Rodrigues, and M. Guizani, "Exploring blind online scheduling for mobile cloud multimedia services," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 54–61, June 2013.

[255] L. Zhou, Z. Yang, J. J. Rodrigues, and M. Guizani, "Exploring blind online scheduling for mobile cloud multimedia services," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 54–61, 2013.

[256] Z. Zhou and D. Huang, "Efficient and secure cata storage operations for mobile cloud computing," in *Proceedings of the 8th International Conference on Network and Service Management*. Laxenburg, Austria, Austria: International Federation for Information Processing, 2013, pp. 37–45.

[257] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation computer systems*, vol. 28, no. 3, pp. 583–592, 2012.