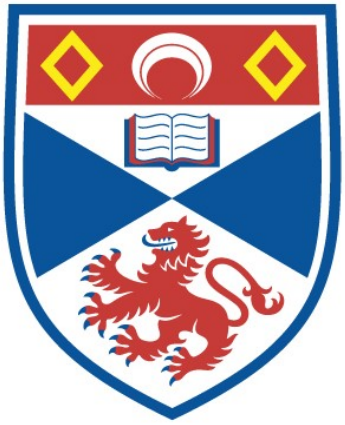
MULTISITE ADAPTIVE COMPUTATION OFFLOADING FOR MOBILE CLOUD APPLICATIONS

Dawand Jalil Sulaiman

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



2020

Multisite Adaptive Computation Offloading for Mobile Cloud Applications

Dawand Jalil Sulaiman

University of
St Andrews

This thesis is submitted in partial fulfilment for the degree of
Doctor of Philosophy (PhD)
at the *University of St Andrews*

January 2020

Abstract

The sheer amount of mobile devices and their fast adaptability have contributed to the proliferation of modern advanced mobile applications. These applications have characteristics such as latency-critical and demand high availability. Also, these kinds of applications often require intensive computation resources and excessive energy consumption for processing, a mobile device has limited computation and energy capacity because of the physical size constraints.

The heterogeneous mobile cloud environment consists of different computing resources such as remote cloud servers in faraway data centres, cloudlets whose goal is to bring the cloud closer to the users, and nearby mobile devices that can be utilised to offload mobile tasks. Heterogeneity in mobile devices and the different sites includes software, hardware, and technology variations. Resource-constrained mobile devices can leverage the shared resource environment to offload their intensive tasks to conserve battery life and improve the overall application performance. However, with such a loosely coupled and mobile device dominating network, new challenges and problems such as how to seamlessly leverage mobile devices with all the offloading sites, how to simplify deploying runtime environment for serving offloading requests from mobile devices, how to identify which parts of the mobile application to offload and how to decide whether to offload them and how to select the most optimal candidate offloading site among others.

To overcome the aforementioned challenges, this research work contributes the design and implementation of *MAMoC*, a loosely coupled end-to-end mobile computation offloading framework. Mobile applications can be adapted to the client library of the framework while the server components are deployed to the offloading sites for serving offloading requests. The evaluation of the offloading decision engine demonstrates the viability of the proposed solution for managing seamless and transparent offloading in distributed and dynamic mobile cloud environments. All the implemented components of this work are publicly available at the following URL: https://github.com/mamoc-repos

Declaration

Candidate's declaration

I, Dawand Jalil Sulaiman, do hereby certify that this thesis, submitted for the degree of PhD, which is approximately 49,700 words in length, has been written by me, and that it is the record of work carried out by me, or principally by myself in collaboration with others as acknowledged, and that it has not been submitted in any previous application for any degree.

I was admitted as a research student at the University of St Andrews in January 2016.

I received funding from an organisation or institution and have acknowledged the funder(s) in the full text of my thesis.

Date 21/1/2020 Signature of candidate

Supervisor's declaration

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of PhD in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date 21/1/2020 Signature of supervisor

Permission for publication

In submitting this thesis to the University of St Andrews we understand that we are giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. We also understand, unless exempt by an award of an embargo as requested below, that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that this thesis will be electronically accessible for personal or research use and that the library has the right to migrate this thesis into new electronic forms as required to ensure continued access to the thesis.

I, Dawand Jalil Sulaiman, confirm that my thesis does not contain any third-party material that requires copyright clearance.

The following is an agreed request by candidate and supervisor regarding the publication of this thesis:

Printed copy

No embargo on print copy.

Electronic copy

No embargo on electronic copy.

Date 21/1/2020 Signature of candidate

Date 21/1/2020 Signature of supervisor

Underpinning Research Data or Digital Outputs

Candidate's declaration

I, Dawand Jalil Sulaiman, hereby certify that no requirements to deposit original research data or digital outputs apply to this thesis and that, where appropriate, secondary data used have been referenced in the full text of my thesis.

Date 21/1/2020 Signature of candidate

Acknowledgements

I would like to acknowledge my supervisor, Prof. Adam Barker, for his patience and invaluable guidance in this journey. Thank you for treating me as a friend and creating such a relaxed environment for our discussions.

My appreciation goes to the head of school, Prof. Simon Dobson, the school manager, Alex Bain, and the systems and front desk office admin members for their support. Also, the libraries of University of Abertay and University of Dundee where I spent long hours to produce this thesis.

I deeply appreciate the help of my current and former PhD colleagues including Dr. Simone Conte for his valuable and continuous feedback on my PhD work and Dr. Ward Jaradat for his great insights and guidance during coffee breaks. I would also like to thank Dr. Khawar Shahzad, Mohammad Ramadan, and Sheriffo Ceesay for the great times and the fruitful discussions. Thanks to Xu Zhu, Xue Guo, Wangjia Yu, and Teng Yu for their company and allowing me to practice my spoken Chinese with them.

I am in forever debt to my parents who provided me with everything I needed to grow and excel. The emotional support and endless love throughout this journey and always believing in me.

I would also like to acknowledge my good friend and mentor, Dr. Graeme Bell for his precious advice, inspiration, and support.

Finally, I want to thank my bonny wife, Simav. Her comfort and encouragement have carried me on a long way through the difficulties in both my PhD candidature and life. This work is for you.

Funding

This work was supported by the University of St Andrews (School of Computer Science). I thank the University of St Andrews for the research training support and facilities to let me pursue my PhD degree.

Table of Contents

List of Figures

List of Tables

List of Codes

List of Algorithms

Chapter 1

Introduction

The number of mobile users exceeded desktop users by the end of 2014 [92]. This growth is expected to continue with the opportunity lying in mobile internet that will add 1.75 billion new users over the next eight years, reaching a milestone of 5 billion mobile internet users in 2025 [141]. From an economic perspective, consumers downloaded over 30.3 billion mobile applications only in the second quarter of 2019 with consumer spend hit nearly $22.6 billion in revenue, up to 20% year over year [8]. Mobile devices have become an essential part of modern life in this new era of mobile computing and Internet of Things. However, there exists a contradiction between the inadequate processing capacity of mobile devices and the users' ever-growing need for better performance and longer battery life. A wide range of applications is now executed on mobile devices, many of which demand high computational power. The tremendous success of mobile technologies is placing severe strains on the underlying resources needed to continue the growth and deployment of new users, new applications, and new services.

Mobile devices are constrained by many limiting factors, including battery life, storage space, and CPU speed. To overcome these limitations, many augmentation approaches have been proposed by researchers, including offloading computation to more powerful servers. Facilitating seamless integration of mobile computing and cloud computing has led to the emergence of a research topic of Mobile Cloud Computing (MCC). Modern mobile devices have become advanced in terms of processing speed, sharper display screens and greater sensors that cause higher energy consumption. Backed by the unbounded resources of cloud computing, MCC can meet the demands of even the most computationally and resource-intensive applications.

Mobile Computation Offloading (MCO) has become a promising method to reduce execution time and save the battery life of mobile devices. The process

involves augmenting execution through migrating heavy computation from mobile devices to high-performance cloud servers and then receiving the results via wireless networks. The constraints of the mobile devices in terms of execution power and battery life make the idea of offloading attractive. Unfortunately, in MCC, offloading to Remote Cloud infrastructure is not always guaranteed to be time efficient and energy conserving [12]. When the network bandwidth is fairly limited, it may be too slow to transmit data between Host Mobile Devices and Remote Cloud servers; When the network status is highly unstable, maintaining a connection to a server might consume more energy than local computation. With continuous growth in the number of neighbourhood mobile and fixed devices, we envision designing environment-aware mobile cloud applications that span across multiple cloud resource levels.

The heterogeneous mobile cloud environment contains different types of computing resources such as Remote Clouds, Cloudlets, and Nearby Mobile Devices in the vicinity that can be utilised to offload mobile tasks. Heterogeneity in mobile devices includes

- Software with multiple versions of phone/tablet OS such as Android & iOS, laptop OS such as MacOS, Windows and Linux). For example, more than 7 different versions of Android are in popular use in 2019 [6].
- Hardware with diverse ARM-based and x86-based low power solutions for phones, tablets, and laptops.
- And technology variations such as different WiFi chip-sets, supported WiFi standards, and power-management approaches.

The current frameworks in the literature lack the automated transparency feature so that the surrounding devices can be detected and the computation offloading take place in a seamless manner [117]. Among other challenges, dynamic environmental changes are one of the most significant facing offloading decision making in mobile cloud applications. Mobile cloud frameworks need to adapt to these changes for efficient task partitioning and high QoS of mobile applications running on end-user devices.

Therefore, this thesis focuses on multisite adaptive mobile clouds that aims to utilize the surrounding service providers and ensure adaptation to different mobile cloud environments. Each mobile device within the shared environment checks whether a task is worth offloading and where to offload its computation among the available external platforms. MAMoC Client is a mobile client library which allows an Android mobile device to offload its tasks (classes or methods) to

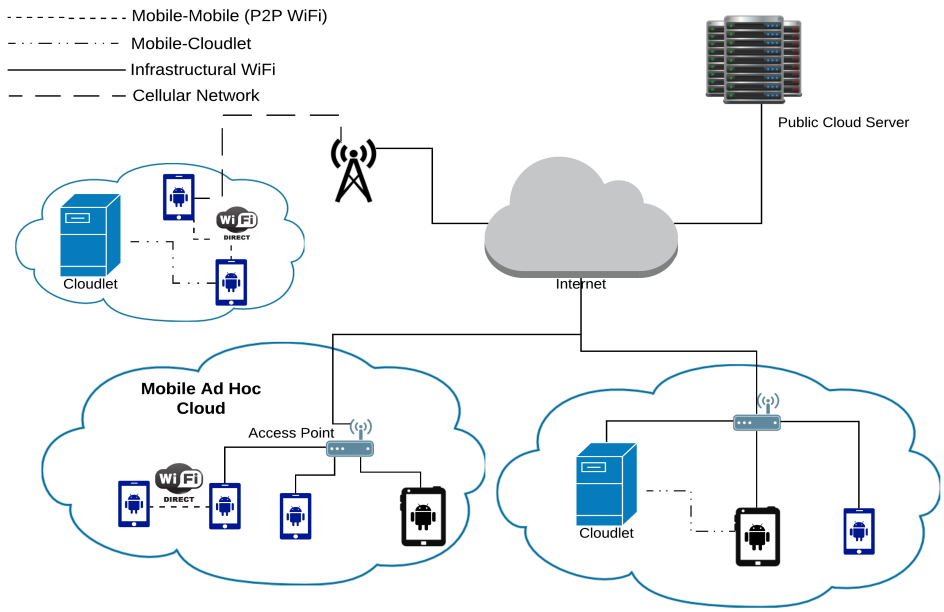


Fig. 1.1 General architecture of multisite mobile cloud systems

other offloading sites including Nearby Mobile Devices running Android OS, fixed edge devices (also called Cloudlets [120]) such as laptops and desktops or Remote Clouds instances such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure datacenter regional servers. MAMoC Server is a set of lightweight containers including a runtime environment deployed to the Cloudlet and Remote Cloud servers.

The general architecture of the framework is depicted in Figure 1.1. The objectives of the proposed solution include performance enhancement in terms of computational time by offloading resource-intensive computations to more powerful external resources, energy efficiency by reducing the computational overhead on the mobile device, context-awareness by making smart offloading decisions considering the associated cost of computation and offloading delay, code reusability by following a highly modular approach, and high adoptability by keeping the adoption of the application model easy for the application providers. The implementation of both the mobile client offloading library [1] and server

[1]https://github.com/mamoc-repos/MAMoC-Client

runtime environment solution [2] are open-sourced. The demonstration of the framework and integration guidelines are also publicly available [3].

The remainder of this chapter describes motivations for studying the challenges of dynamic mobile cloud environments in Section 1.1. Section 1.2 describes the research hypotheses of this thesis. Section 1.3 presents an overview of the contributions of this thesis. Section 1.4 lists the peer-reviewed papers that were published during the course of conducting this work. Finally, Section 1.5 outlines the structure of the rest of the chapters of this thesis.

1.1 Motivations and Challenges

This thesis is motivated by the notion that offloading parts of a mobile application to multiple sites can improve the performance and reduce the overall energy usage of the mobile device. The following motivations and challenges need to be considered in designing an adaptive and multisite offloading system:

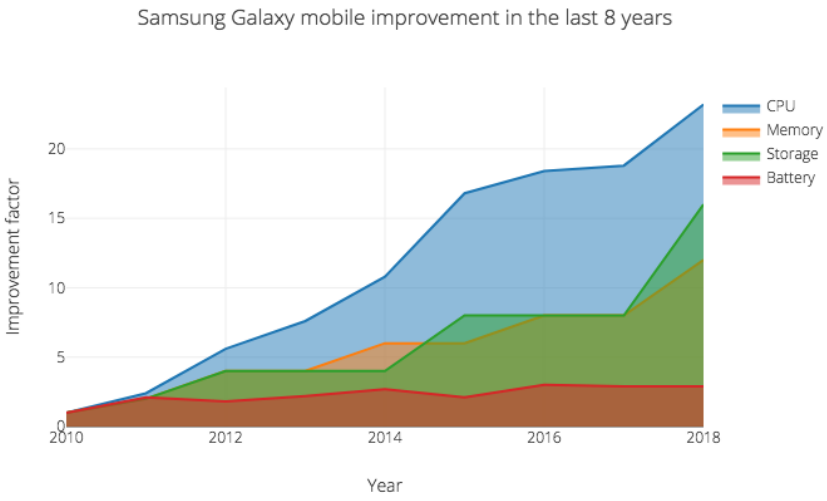


Fig. 1.2 The hardware advancements of Samsung galaxy smartphones

Extending battery life

A study established that the most commonly expressed criterion for choosing a mobile phone in 2018 was the battery life, with over 46% of users nominating it as their "most important aspect" [132]. Poor battery life is among the many problems that confront mobile devices but the most

[2] https://github.com/mamoc-repos/MAMoC-Server
[3] https://github.com/mamoc-repos/MAMoC-Demo

apparent one. The fact that the development of battery life is far behind other hardware advancements has been a struggle for device manufacturers in the last several years. As an example, it can be observed in Figure 1.2 for the Samsung galaxy family smartphones that the CPU performance has improved about 23 fold, memory storage about 12 and 16 fold respectively while the improvement in battery life is merely 2 to 3 fold.

This trend can be identified across all mobile devices in the market. As an example, the hardware specifications of the first iPhone and HTC smartphones with the latest releases of them in 2018 are listed in Table 1.1. It is evident that the battery capacity of iPhone smartphones has only doubled while the HTC battery is at most tripled. This is incomparable with the exponential growth of the other hardware parts of both smartphones.

Specification	2008		2018	
	iPhone 2	HTC Dream	iPhone X	HTC U12+
CPU	412Mhz	528 MHz	2.39GHz Hexa-core	2.45 GHz Octa-core
RAM	128MB	192MB	3GB	6GB
Storage	8GB	256MB	256GB	128GB
Battery	1400 mAh	1150 mAh	2716 mAh	3500 mAh

Table 1.1 iPhone and HTC hardware advancements in the last 10 years

It is benchmarked that most of the battery energy is consumed with heavy computations, which causes overheating [160]. It is shown that the CPU power consumption to total power consumption can be up to 40% in the gaming applications [24]. Therefore, through offloading computation from mobile devices to external servers, we can decrease the load on the device and reduce the energy consumption of the heavy tasks which results in extending the battery life.

Availability of multiple service providers

The last few years have witnessed unprecedented growth in the number of mobile devices. People are becoming more reliant on their smartphones, and the cost of purchasing these devices has reduced in emerging markets. Globally, Cisco indicates that there will be 12.3 billion mobile-connected devices accumulating for around 1.5 mobile devices per person by 2022 [32] as shown in Figure 1.3. There are a vast amount of mobile cloud offloading systems that have been developed by different researchers. As described in Chapter 3, the majority of them use a single-site offloading, i.e., offloading

application parts from the mobile device to a single server. Moreover, the existing works do not provide a formal framework to include the overall offloading process; Rather, they conduct numerical results of their reference architectures through simulations. The vision of mobile computing among heterogeneous converged networks and multiple computing devices can be achieved by designing resource-efficient environment-aware mobile cloud systems.

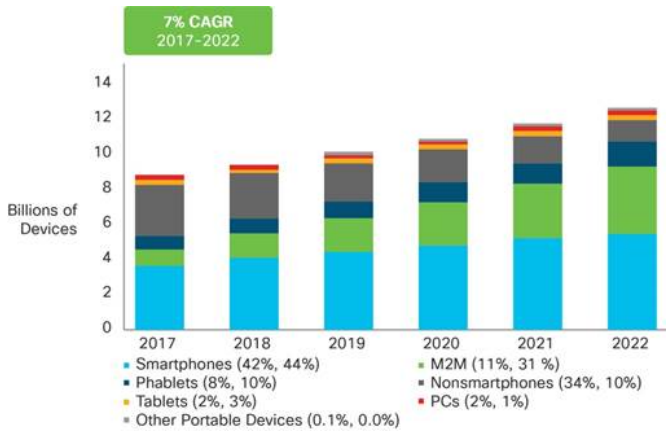
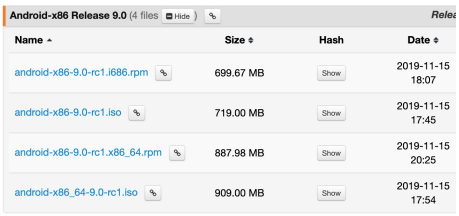


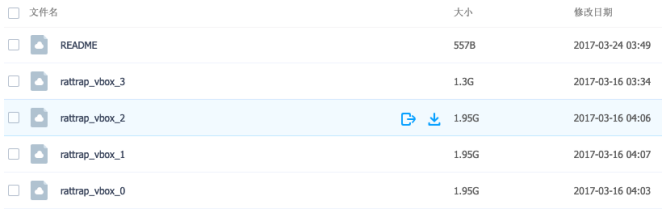
Fig. 1.3 Global Mobile Devices and Connections Growth [32]

Dynamic nature of mobile cloud systems

The connection between mobile users and external cloud resources is not guaranteed. The non-persistent connection is a key factor distinguishing mobile cloud with conventional cloud systems. The essence of random unavailability of wireless connections in mobile cloud environments makes consistently beneficial offloading difficult. The presence of intermittent connections may fail the offloading request from the user, which causes the user to execute it locally or re-transmit the job again. Mobile device movement, jointly with unstable wireless networks, can lead to service failures in mobile cloud systems regularly. The performance of a certain type of cloud resources such as Mobile Ad-hoc Cloud (MAC) (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 providing a reliable mobile cloud service. Other examples of dynamic changes in the system include devices joining/leaving the cloud, battery running low, temperature rising too high, OS throttling the CPU, etc. Therefore, it is imperative to consider these changes before making computation offloading decisions at all times to avoid undesirable application performance.



(a) The file sizes of Android-x86 VM images [104]



(b) The file sizes of Android-x86 container images [155]

Fig. 1.4 Android-x86 VM and container image file sizes

Deployment complexity of runtime environments

To design a reliable mobile cloud system, provisioned offloading servers with high availability need to be present to serve the offloading requests from mobile devices. Unfortunately, these solutions have not been implemented yet worldwide [51]. Many companies lack enough equity to cope with the expense of deploying small clouds at multiple base stations or other locations at the edge of the network. There is still an ongoing difficulty when trying to set up an offloading runtime environment based on Android server-side OS (Android x86) to serve offloading requests. Figure 1.4a shows the latest Android x86 image files of which the 64-bit OSs are over 800 Megabytes in size. Figure 1.4b shows an effort by researchers proposed in a paper [155] to run a lightweight version of Android on containers rather than Virtual Machine (VM)s. However, it is apparent that the size of the files is still too large. An investigation of reducing the size and simplifying the deployment of these solutions is required as well as the ease of managing the servers remotely.

Platform heterogeneity and inter-operability

The diversity of hardware architectures and operating systems in mobile devices makes it difficult to design a uniform mobile cloud system. In an "always offload" strategy, transmitting a large amount of data for remote execution may consume more time and energy than running it locally on the device itself. It would be more beneficial to acquire the same processing power in nearby mobile or fixed devices in the ad hoc network through parallel execution and low latency network transmission. Therefore, it is essential to develop intelligent decision-making strategies for selecting the right candidates in different scenarios. Network heterogeneity also needs to be taken care of in scenarios where mobile users switch to cellular networks when there is no WiFi network available. The energy consumption and

transmission speeds are different in these assorted types of wireless network interfaces. There is a need for a framework that can provide an abstraction of these heterogeneous, resource-constrained devices in order to allow for seamless development, deployment, and management of mobile applications.

Transparency and automation mechanisms

Even after enabling offloading between different devices with various computational capabilities; It is essential to hide these interactions from the user. Transparency is the ability to implement software without the need for manual program modification [83]. It is important for mobile phone users to have transparency and choice [44]. In offload-enabled mobile applications, invoking instrumentation transparently and selectively each time the application executes is a challenging task. Automated refactoring processes are needed to make application components follow an appropriate adaptation strategy.

1.2 Research Hypotheses

This thesis investigates two central hypotheses regarding optimising mobile devices through mobile computation offloading techniques. The first one is related to the goal of improving the performance of mobile applications by offloading parts of them while minimising the incurred energy consumption of the host mobile device. The second is concerned with facilitating the deployment of offloading service providers and automating the process of determining offloadable tasks. These two hypotheses are considered throughout this thesis and examined in the evaluations.

H1 **Mobile devices can be seamlessly leveraged with all the surrounding sites including nearby mobile devices and edge servers as well as distant cloud resources. Moreover, the decision to offload a task and identify the most optimal candidate for single-site offloading and the most optimal ranking of the candidates for multisite offloading scenarios can be taken adaptively.**

We are surrounded by a vast amount of both mobile and fixed devices. The number of smart device candidates increases every day. It is depicted that on average, a person would have two devices starting from lower-end devices such as smartwatches, wristbands, and smartphones all the way to higher specification computing devices including tablets and laptops [32]. Discovering nearby devices and performing offload operations can be a

simple task between two homogeneous devices but it is not a primitive task with heterogeneous devices which have diverse hardware capabilities and software programs. Accordingly, a lightweight service discovery technique for seamless device integrations are needed to achieve that. The framework should provide an environment for homogeneous application development in such heterogeneous environments. The framework should further make it simple to offload parts of an application either to Nearby Mobile Devices or to Cloudlets and Remote Clouds to meet the application's requirements. The framework should also support simple development, configuration and deployment tools to facilitate the life-cycle management of applications.

The design of algorithms for adaptive application deployment and configuration are essential for seamless mobile offloading. To decide which deployment and configuration are optimal given a certain context at runtime, the decision engine with the help of the dynamic profilers should be capable of deciding whether the task is better off being executed locally on the device or send it to the external site(s) for remote execution. The solver then needs to check the availability of the sites and output a ranking among them with multiple criteria such as the execution speed, bandwidth, and availability of the sites. Decision methods and fuzzy logic for solving this optimization problem can be used when different optimisation goals are considered, such as minimising execution time and energy usage.

H2 **The process of deploying runtime environments for serving offloading requests from mobile devices can be simplified. It can farther be utilised to automatically identify offloadable tasks in unmodified mobile applications when manual task annotation is infeasible.**

One of the bottlenecks of computation offloading solutions is setting up service provider runtime environments in the offloading sites. Previous MCC studies have used VMs to host mobile operating system images in the cloud. Hardware virtualization allows different guest operating systems to coexist in the same physical server. The main drawbacks of using VMs are the long startup time and high virtualization overhead. Heavyweight VM solutions cannot meet these requirements. Though pre-starting the VM can reduce its startup time, it would inevitably incur a high resource cost due to numerous offloading requests. Moreover, it cannot work as a service provider for different mobile devices with various architectures. Because of the aforementioned heterogeneity in mobile devices, different runtime

environments need to be deployed for handling service providers. Decreasing the boot-up time and deploying a uniform and loosely coupled runtime environment which accommodates to serving offloading requests from mobile devices with different capabilities can greatly simplify this process.

During the evolution of a mobile application, the developer can manually annotate the classes or methods that are compute-intensive to be executed remotely. However, to support the existing mobile applications in app stores, an automation process needs to be designed to enable the offloading mechanism even without access to the original source code. Static analysis and bytecode injection tools can be used to allow us to overcome the difficulty of supporting arbitrary mobile applications where it is impractical to solicit developers for manual annotation.

1.3 Contributions of This Research

1. A comprehensive review of the mobile cloud architectures and a derived taxonomy of multisite MCO research works.

2. An adaptive task offloading algorithm for individual mobile devices to make offloading decisions based on the context changes in dynamic mobile cloud environments.

3. An offloading site ranking system based on Multi-Criteria Decision Methods (MCDM) and fuzzy logic to select candidate offloadees and achieve different optimization goals using group decision-making concept.

4. An overview and discussion of the design principles and challenges of implementing MAMoC, a system framework to enable mobile task offloading for heterogeneous mobile cloud systems that can be used as a simple programming model to build mobile cloud applications.

5. A loosely coupled, router-based, containerized, and lightweight runtime environment for serving the mobile offloading requests on the heterogeneous offloading sites for providing a more scalable and reliable offloading service.

6. Seamless support for arbitrary mobile applications by determining the set of tasks that can be offloaded without any constraints. An automated task annotation methodology using static analysis and code instrumentation techniques is proposed when the manual annotation is infeasible.

1.4 Publications

The work conducted in the course of my PhD resulted in publishing a number of peer-reviewed papers in international conferences. The major works including the framework design, implementation, and evaluations presented in **??**, **??**, and Chapter 6 are extracted from the publications with necessary modifications. The papers are listed as follows:

- Dawand Sulaiman and Adam Barker, "MAMoC-Android: Multisite Adaptive Computation Offloading for Android Applications", 2019 7th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Newark, CA, USA, 2019, pp. 68-75. [137]
- Dawand Sulaiman and Adam Barker, "MAMoC: Multisite Adaptive Offloading Framework for Mobile Cloud Applications" 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, 2017, pp. 17-24. [136]
- Dawand Sulaiman and Adam Barker, "Task Offloading Engine for Heterogeneous Mobile Clouds." In Proceedings of The 8th EAI International Conference on Mobile Computing, Applications and Services. ACM, 8th EAI International Conference on Mobile Computing, Applications and Services, Cambridge, United Kingdom, 2016. [135]

Additionally, a survey paper will be extracted from the contents of Chapter 3 and will be published in a Journal.

1.5 Organisation of The Thesis

The remainder of this thesis is structured as follows:

- Chapter 2 provides a detailed overview of mobile cloud architectures and key techniques including computation offloading and related research area.
- Chapter 3 explains the multisite offloading in the area of MCO and collects the literature work. Moreover, state-of-the-art on multisite MCO is provided in the form of a systematic review.
- Chapter 4 defines the system requirements and models and formulates the problem of offloading to multiple offloading sites in terms of execution time and energy consumption. It also explains the offloading policy and decision making algorithms that are core parts of the offloading decision engine. Additionally, it describes MCDMs that are used to evaluate and rank the candidate offloading sites.
- Chapter 5 discusses the design assumptions and the components of the mobile offloading framework. It also describes the different phases of the task execution workflow. Moreover, it provides an overview of the communication mechanisms used in the service discovery module for handling communications between Host Mobile Device and offloading sites. The chapter also highlights interesting aspects of the implementation of software components within the framework in both Host Mobile Device and offloading sites.
- Chapter 6 evaluates the framework through four sets of real-world experiments, evaluates the proposed design requirements, analyses and discusses the results, and presents the evaluation limitations.
- Chapter 7 presents a summary of the thesis, revisits the hypotheses and the contributions, and proposes future research directions.

Chapter 2

Background

This chapter presents the background and context of mobile cloud architectures and setting the ground for next chapter to review the existing work in research and development of multisite offloading approaches. It begins with the background technologies of the different mobile cloud architectures and a comparison of the different paradigms in Section 2.1. Then, it describes computation offloading with a focus towards the two important aspects, including adaptability and multisite offloading features in Section 2.2. Finally, Section 2.3 describes some of the main technologies and communication mechanisms used in the field of MCC research.

2.1 Mobile Cloud Architectures

The global mobile data traffic handled by Cisco has grown 4,000 fold over the past 10 years and almost 400 million fold over the past 15 years. The data traffic grew 74 per cent in 2015 reaching 3.7 Exabytes per month at the end of 2015, up from 2.1 Exabytes per month at the end of 2014 [32]. This is mainly because of modern data and computation-intensive mobile applications that are resource hungry and cannot be run locally. To cater to ever-increasing requirements for storage and network resources, efficient data distribution and optimization techniques need to be adopted. The result of the vast growth in the number of mobile devices along with transferring data and compute-intensive applications to the cloud has led researchers to search for more effective and intelligent ways to create this two-way relationship.

The current MCC trend has set the focus on the ubiquity of computation. However, the main current cloud platforms still confine the servers into datacenters, which are far from the users. Distance leads to increased energy consumption in the network, higher utilization of the broadband Wide Area Network (WAN) and

poor client experience, especially for latency-critical applications. As a solution, distributed architectures try to locate computation among different sites. These distributed cloud architectures that bring data closer to the client are emerging as an alternative to centralised ones.

In this section, an overview of the most researched areas in mobile cloud studies is discussed, starting with MCC and its three different models. The more recently emerged post cloud computing studies which are Edge Computing and Fog Computing and their approaches are explained regarding mobile device augmentation. Finally, the similarities and differences between these paradigms are demonstrated with real-world use cases and supporting statistics.

2.1.1 Mobile Cloud Computing

Advancements in mobile device technologies have led to the production of more complicated and interactive applications such as face recognition, real-time video, healthcare monitoring, Augmented Reality, and Virtual Reality mobile applications. The common feature between these applications is that they all need high processing power and fast and near real-time response time. Due to these requirements, the concept of computation offloading from mobile devices to resourceful cloud servers has been introduced as MCC. The complicated computation-intensive modules of the mobile applications can be easily executed with a fast response time on the powerful remote surrogates [77]. However, the performance of offloading is highly affected by the available bandwidth and latency [129]. Mobile computation offloading will be explained in detail in Section 2.2.

MCC is an emerging paradigm that encompasses mobile computing, cloud computing, networking and virtualization. It is well studied in the literature. There are many extensive MCC surveys in the literature [44] [128] [38] [76]. MCC can support mobile devices through either computation augmentation which is about leveraging powerful external resources to execute the heavy parts of the application or storage augmentation for using the excessive data storage capabilities of the cloud [163].

Because of high latency and low bandwidth issues of remote cloud servers in traditional MCC studies, researchers have suggested using one-hop away powerful surrogates to the mobile devices in the form of Cloudlets which is discussed in Section 2.1.1.1. As smartphones and tablets gain more CPU power and longer battery life, the meaning of MCC gradually changes. Instead of being fully hooked to Remote Cloud, a number of Nearby Mobile Devices can be used to coordinate and distribute content in a decentralized manner discussed in Section 2.1.1.2.

2.1.1.1 Cloudlet

Cloudlet is viewed as a small-scaled cloud having a cluster of computers that is well-connected to the Internet serving mobile devices in close proximity. In this way, we manage to overcome the high latency problem faced in the traditional MCC approaches, which depend on Remote Cloud servers only. The idea of using cloudlets (surrogates) as near fixed devices was introduced in [120] in which the mobile device offloads its workload to a local cloudlet comprised of several multi-core computers with connectivity to the remote cloud servers. Satyanarayanan [119] uses four futuristic scenarios as case studies for the facts on the architectural evolution of mobile cloud computing from a 2-tier (mobile device - cloud) to a 3-tier (mobile device - cloudlet - cloud). Several approaches to this design are mentioned in this paper, such as VM synthesis and code offloading. Alongside solving latency issues, he sheds light on other value propositions of cloudlets such as the bandwidth, crowd-sourcing, privacy and security, and availability.

From an energy point of view, offloading to cloudlet saves energy because of access to the short-range wireless connection [48]. Sending and receiving data not only claims large shares of wireless bandwidth but also drains the battery of mobile devices. It is shown that despite several new power-saving mechanisms of 4G/LTE cellular data; it requires 23 times more power than WiFi and it is less power efficient than 3G [61]. In some works, the resource-intensive application tasks are first offloaded to the cloudlet, and then later to nearby devices in the LAN or remote cloud resources if needed [147]. In case the connection to cloudlet service is interrupted, the offloaded tasks can be retrieved back and send them to be executed on alternative avenues.

The main goal of using cloudlets is to be able to reduce the response time in order to meet the needs of some latency-sensitive applications [49]. The advantage of being close to the user allows us to achieve this goal but does not provide us with the same computation and storage capabilities as those of the remote cloud. Thus, its computing capacities are limited to certain services. In addition to its proximity to the users, Cloudlet also has the advantage of being exploited by mobile users who do not even have an Internet connection [120].

2.1.1.2 Mobile Ad-hoc Clouds

The concept of Ad-hoc Cloud Computing has been discussed in [91]. According to [44], apart from offloading to central clouds, there are two other definitions of MCC. The first is where some mobile devices act as cloud resource providers forming a Peer-to-Peer (P2P) network. In this model, the mobile devices in the

local vicinity and other stationary devices (if available) would create an ad-hoc network which can be accessed by other mobile devices in order to run their applications. Theoretically, this model allows offloading the tasks to the mobile devices that form the virtual resource cloud. Besides, latency is also reduced, since the mobile users only have to access the virtual cloud resource instead of traversing lots of hops to get to the Remote Cloud.

MACs are about leveraging the computational capabilities of the surrounding mobile devices by having them as resource nodes. It is also known as Mobile Device Clouds (MDC) [95], Mobile Edge Clouds (MEC) [40] [14] [45], and Virtual Computing Provider for Mobile Devices [62]. The most well-known works at accomplishing resource sharing and data distribution among mobile devices in the vicinity include Hyrax [90], Scavenger [75], Serendipity [124], and Cirrus [122].

2.1.2 Mobile Edge Computing

In order to reduce latency between end-users and service providers, without dropping the main benefits of MCC, Mobile Edge Computing (MEC) has emerged as a new paradigm where the functions that have been traditionally located in a remote datacenter are called back to locations nearer to the user, e.g., networking devices in the access network with spare or added computational capabilities. MEC resources may offer more limited capacities or reliability guarantees than core resources but are geographically spread closer to the end-users, providing smaller network latencies [81].

MEC is the new evolution of mobile networking by providing computation capabilities in base stations. Theoretically, data would only need to travel a few miles between customers and the nearest cell tower or central office, instead of hundreds of miles to reach a cloud datacenter. Mobile edge servers are co-placed with the mobile network base station at the edge of the mobile network. Mobile edge hosts in network operators are connected to the mobile core. User equipments are connected to hosts that contain local VMs through API endpoints. The white paper proposed by the European Telecommunications Standards Institute (ETSI) [60] introduced the MEC concept with its reference architecture and application scenarios. MEC is widely regarded as a promising paradigm to enable mobile terminals to enjoy the abundant wireless resource and vast computation power ubiquitously [86].

Cloudlet can be considered as an edge server for their common objectives to bring computation closer to the end-user [59]. The authors at [81] explain the several related computing paradigms to Edge Computing including Content

Delivery Networks (CDN) [144] and P2P computing, among the others. There are also attempts to combine the concept of Cloudlets and Edge Computing in designing future mobile cloud systems [64]. Similar to cloudlets, the MEC system has several advantages including low latency, content caching, traffic monitoring, local aggregation of information, and local services. It should be noted that the ultimate goal of MEC is to increase the bandwidth, reduce the latency and jitter, and provide Quality of Service (QoS) for mobile apps. There are many use cases of MEC, such as CDNs, smart grids, augmented reality, and traffic management [1].

2.1.3 Mobile Fog Computing

Fog Computing was first proposed by Bonomi et al. [17] that describes it as a scenario where a huge number of heterogeneous ubiquitous and decentralised devices communicate and potentially cooperate among them and with the network to perform storage and processing tasks. These tasks can be for supporting basic network functions or new services and applications that run in a sandboxed environment. The rise of the Internet of Things (IoT) popularity made Fog Computing interesting to researchers and practitioners. In IoT, a huge amount of sensors are deployed everywhere. For low-capacity sensors, the best place to process data from these sensors should not be far away cloud servers [87]. Instead, devices such as routers and switches are better choices.

Mobile Fog Computing (MFC) is considered as the complementary of Fog Computing for providing low latency mobile services [58]. Mobile Fog proposed by Hong et al. [56] is a high-level programming model for latency-sensitive and large-scale future applications over heterogeneous devices. Applications can invoke event handling and function calls by the model. The model is evaluated with a simulated use case of a vehicle tracking application where the traffic cameras are used to help police identify and track certain vehicles. The compute-intensive functions of the application, such as license plate recognition, video streaming are all performed on the intermediate fog nodes before the data are sent to Remote Clouds.

Fog computing and Edge Computing have a number of similarities. Although, Fog Computing mainly focuses on communication optimization at the infrastructure level, while Edge Computing manages the computing needs and network demand of both end devices and infrastructure, including the collaboration among end devices, edge servers, and remote cloud.

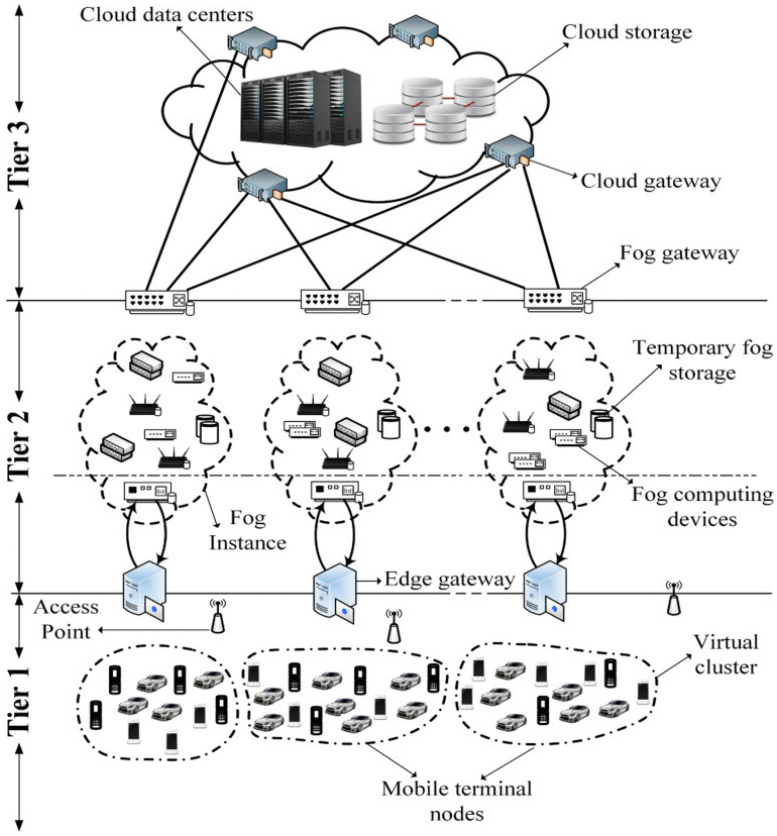


Fig. 2.1 Fog computing architecture [118]

2.1.4 Comparison of Mobile Cloud Architectures

In this section, the different mobile cloud approaches are compared in Table 2.1. The evaluation is based on the following properties, which have been chosen due to their effects on the Quality of Service of mobile applications:

Latency: This property evaluates how fitting is the system for latency-critical applications. In some mobile applications such as Augmented Reality, users request low latencies to avoid having slow frame loading, which affects the performance. As shown in Table 2.1, MCC solutions that offload to Remote Cloud servers have a high latency which does not make them ideal candidates for low latency applications. The delay-sensitive mobile tasks are ideally offloaded to the cloudlets or nearby devices. On the other hand, differences between Edge Computing and Fog Computing are dependent on the location of the edge servers to the mobile user as they are confined to the edge infrastructures.

Distance: This depicts the physical distance of the infrastructure providing the service to the mobile device. This is closely related to the latency due to the

number of network hops that are being introduced further the mobile device gets to the servers. Nearby local devices are usually within 10-20 meters away, while the Cloudlets and edge servers can be as far as a kilometre to Host Mobile Device.

Deployment: The cloud service providers such as AWS and Azure are usually the sources of Remote Cloud servers. To resolve latency issues, fixed surrogates can be deployed in the vicinity to be a one-hop away from the mobile device in the form of Cloudlets. These can also be in the form of macro-datacenters, small clouds, femtocells, etc. MEC servers are deployed in Radio Access Networks which can provide fast services to mobile users.

Computation Power & Storage Capacity: The abundance of resources at cloud data centers and the elasticity feature of Cloud Computing allow for ample computation power and storage capacity. Even though cloudlets and nearby devices solve the latency issues of far clouds, their computation power is rather limited. Edge and Fog Computing use a distributed set of powerful servers deployed in the network edges to provide powerful services to mobile users.

Communication Medium: Mobile devices connect to different service providers using different wireless technologies. Since the public cloud infrastructure is accessed through the Internet, both Wi-Fi or cellular technologies can be used to establish connections between them. Cloudlets are deployed in Local Area Network (LAN) so they can be connected to devices in the local network. Traditionally, Bluetooth has been the most popular choice for Device-to-Device (D2D) communications. However, there are multiple new approaches to achieving nearby peer connection establishments, such as WiFi-Direct and ZigBee [19]. The Edge servers are mainly deployed in network servers, so cellular technologies such as 3G/4G are the common network interfaces to connect to them.

Architecture: The architecture tier is a physical structuring mechanism for the system infrastructure in which the computation levels depend on the existence of devices and communications. The most common architecture is when the mobile device is only connected to a single Remote Cloud server that forms a 2-tier mobile-cloud architecture. Cloudlets are used as middle managers to form a 3-tier mobile-cloudlet-cloud architecture for enabling the cloudlet to be closely connected to both the mobile device and the cloud. The MAC devices are normally in the same architecture tier. Meanwhile,

MEC and MFC do not have a standard architecture so devices in different cloud resource levels can be included such as smart routers, nearby mobile devices, forming a multi-tier architecture as shown in Figure 2.1.

Availability: Service availability is one of the most important factors affecting responsiveness and energy consumption in mobile device augmentation scenarios. Cloud providers guarantee high availability of services through a Service Level Agreement. Although cloudlets are deployed close to mobile users, in cases of service churns and multi-user requests, the limited server might not be able to serve all the users in the same way as remote cloud servers. As for MAC, because of the mobility feature of nearby mobile devices, it makes them highly volatile and could become non-available on the fly [93]. MEC and MFC solve the issue of mobility using horizontal and vertical handoff strategies to avoid service disruptions for mobile users [86].

Use Cases: The applications running on Cloudlets and Nearby Mobile Devices are typically time critical and require very low computing and communication latency in an environment with limited bandwidth, and limited computing power. For example, applications such as Linpack (computation-intensive), 3D Car Racing (interaction-intensive) and Chess (computation & interaction-intensive) [147]. MEC is actively used in smart city planning and video surveillance scenarios [86] while MFC conducted studies are mainly targeted towards IoT applications [17].

Operators: Cloudlets are mostly deployed by an individual or a local business such as a coffee shop owner [49]. Since MAC is developed in an ad-hoc fashion, it can be performed locally by an individual or a number of individuals within an organization. The network providers have added powerful edge servers in their base stations to enable MEC [88]. Fog nodes can either use the edge servers in base stations or cloud servers in cloud service providers [158].

Table 2.1 Comparison of mobile cloud related paradigms

Property	MCC			MEC	MFC
	Core	Cloudlet	MAC		
Latency	High	Low	Low	Low	Relatively low
Distance	Far	Close	Very close	Close	Relatively close
Deployment	Data centers	Fixed surrogates	Nearby devices	Network edge (RAN)	Fog nodes
Computational power & Storage capacity	Ample	Limited	Very limited	Fair	Ample
Communication medium	WiFi/Cellular	WiFi	WiFi/Bluetooth	Cellular	WiFi/Cellular
Architecture	2-tier	2-tier or more	2-tier	2-tier or more	3-tier or more
Availability	High	Average	Low	Average	High
Use cases	Social networking, health care	Immersive (AR & VR) apps	Disaster relief, privacy-preserving local processing	Video surveillance, video caching, traffic control, health monitoring, AR	IoT, Connected vehicles, smart city, smart delivery
Operators	Cloud service providers	local businesses	self-organized	Network infrastructure providers (RAN-based)	Cloud service providers and network infrastructure providers

MCC: Mobile Cloud Computing, **MAC**: Mobile Ad-hoc Cloud, **MEC**: Mobile Edge Computing, **MFC**: Mobile Fog Computing

2.1.5 Discussion

The centralised processing model uploads computation and data to the cloud datacenter through the network and leverages its ample power to solve the computing and storage problems of resource-constrained devices. However, traditional cloud computing has several shortcomings, including high latency and insufficient bandwidth, which leads to high energy consumption for limited devices.

While Cloudlets are mostly managed by individuals and can be deployed at any appropriate location such as coffee shops and university campus buildings [49], MEC servers are owned by mobile operators and need to be located near the base stations in order to provide access to the mobile network users over Radio Access Network (RAN). This helps operators to increase service quality through effective mobility management besides the benefits of utilising cloud-like resources at the edge [86] to perform computation offloading on edge [89]. Computation offloading techniques formulated for cloud technologies are now common to edge and fog computing paradigms. Many researchers have pursued the idea of offloading computation to multi-site or heterogeneous resources at once [86].

A notable difference between Cloudlets and both MEC and MFC is in the virtualization technology. Most of the cloudlet research work use VM technology for virtualization [147], while Edge and Fog studies consider other types of virtualization techniques [88]. Another difference is that MEC functions in a stand-alone mode where the tasks are processed at the edge of the network. Cloudlets can function in either stand-alone mode or connected to a cloud. Meanwhile, MFC is designed as an extension to the cloud, so it is mainly in connected mode. Moreover, Cloudlets are mainly designed for mobile offloading solutions (e.g. wearable cognitive apps [26]) while MEC aims at any application that is better provisioned at the edge. Fog-enabled applications can span the public cloud and network edge. This is demonstrated in the firefighting application use-case in [158].

To examine the popularity of these different paradigms, we use a tool [134] to find the published academic papers for each of the aforementioned techniques. As we can note from Figure 2.2, MCC has been well researched and kept the pace from 2012 until recently where Edge Computing (EC) and Fog Computing (FC) papers have gained more popularity. This could be due to the more modern approaches of EC and FC to include a wider spectrum of devices (e.g. IoT devices) instead of only focusing on smartphones which most of the MCC research has been centralised around it. Since 2015, the number of papers related to Edge/Fog Computing has grown tenfold. EC has entered the rapid growth period. Researchers have also been investigating MEC which started around the same time

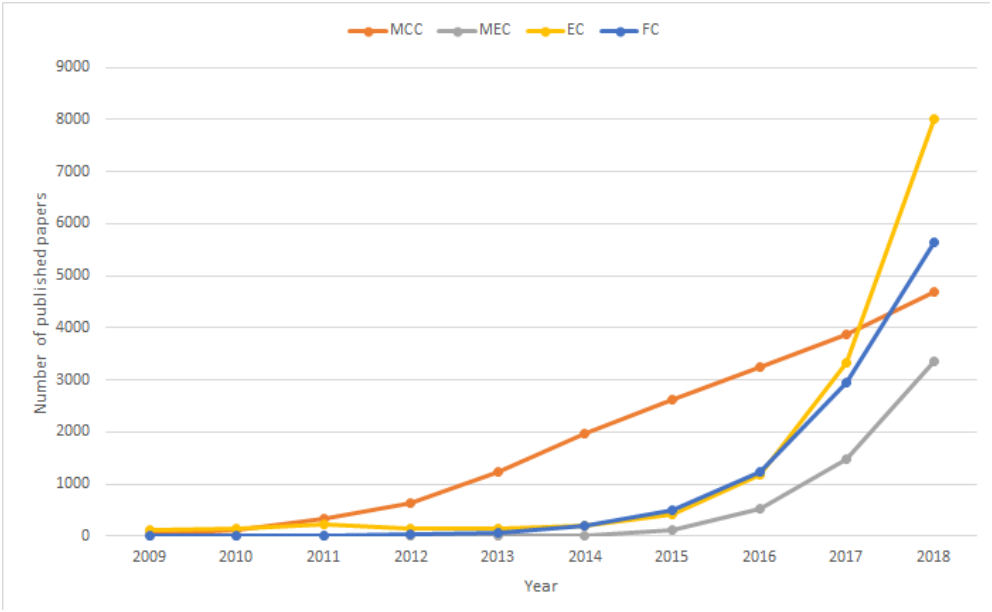


Fig. 2.2 Number of published papers by year for MCC, MEC, EC, and FC

the EC attracted the attention of academia and industry. The pace of it might not keep up with EC and FC as the name was currently changed to Multi-access Edge Computing (MEC) to include a wider variety of devices [138].

2.2 Mobile Computation Offloading

Offloading computation from a weaker device to a more powerful server is not a new concept. The first attempt to conserve the energy consumption of laptops was introduced by offloading larger tasks to fixed surrogates [114]. Then, the concept of cyber-foraging was termed in [10] for offloading parts of applications from mobile devices to nearby discovered servers. Computation offloading either fully or partially migrate the resource-heavy parts of a mobile application to nearby resource-rich surrogates such as computational clouds. Computation offloading not only conserves the usage of local resources such as memory, battery, and storage but also enables execution of computation-intensive applications in mobile devices.

MCO is a mechanism to enable mobile devices to run mobile applications that require extensive computation and communication [84]. It extends battery life by moving the computation-intensive portions of the application to external resources with greater computation power. In addition, it decreases latency and communication costs by using single-hop proximity to nearby devices [49]. This improves the user experience, especially in highly interactive applications that

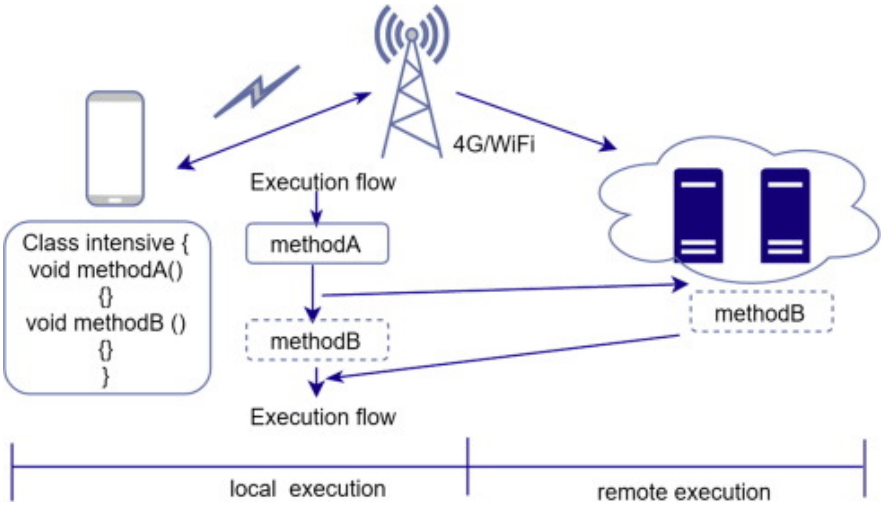


Fig. 2.3 Computation Offloading Process Overview [4]

require high bandwidth and low latency. Figure 2.3 illustrates the environment that supports computation offloading. In this overview, the mobile device decides to offload method B to a cloud server or a powerful machine. The cloud here provides virtual computation resources to run the offloaded components. It can be another device, a server or cluster in a nearby location or a virtual server in the public cloud datacenters.

To enable computation offloading, an offloading client needs to be running on the Host Mobile Device and an offloading server in the external resources. They need to be able to communicate for coordinating the offloading operation. After the application starts executing, the invoked task that has been identified as offloadable will be passed to the offloading client. The offloading client can either follow the "always offload" scenario where every invoked task will be offloaded, or an offloading runtime decision is made (based on a comparison of local and remote execution times or energy consumption) before transferring the task to the offloading server. Any other metadata (invocation parameters, resource files, etc.) will also be provided to the offloading server. The offloaded component starts executing and communicates directly with the mobile app for sending back the results of the remote execution.

Since offloading migrates computation to a more resourceful computer, it involves deciding whether and what computation to migrate. A vast body of research exists on offloading decisions for (1) improving performance and (2) saving energy. The offloading decision making is the bottleneck and the most prominent part of the mobile computation offloading process. This offloading decision making process is being affected by more than a factor [76]. The four

aspects of offload decision making are studied in mobile cloud offloading scenarios [152].

When to offload: an optimal offloading decision has to be made at the right time to offload under different conditions of the device, such as available bandwidth, amount of data to be transferred, and energy to execute.

What to offload: in dynamic mobile environments, full offloading is not always beneficial, so choosing the right component to offload by splitting a specific application into local and remote parts is also important.

Where to offload: since mobile devices are surrounded by multiple offloading sites, it is essential to find the optimal site in which the computation to be offloaded under different cloud resource conditions.

How to offload: Due to the heterogeneity of communication networks available between the mobile device and cloud resources, the right path to offload needs to be determined.

2.2.1 Adaptive Offloading

A system is considered being adaptive if it is designed to continuously monitor its environment and then modify its behaviour in response to changing environmental conditions [15]. Mobile computation offloading systems are affected by wireless network characteristics and the capabilities of available external resources. The workload also varies depending on the applications that are running on the mobile device. For designing an effective offloading system, a range of environmental variables such as bandwidth variation, intermittent connectivity and user expectations need to be considered [15]. User expectations need to be incorporated into the architecture of the offloading system. The different parameters that are set by user preferences need to be defined in the deployed mobile application.

In mobile computation adaptive offloading, the system needs to implement sufficient and accurate change detection mechanisms to detect the changes in the dynamic environment. It has to be responsive to the changes and ensure a fast, accurate, user transparent, and low overhead techniques for change detection. The quality-aware opportunistic response mechanism can be implemented by employing adaptive techniques for using the available resources and by implementing high-quality degradation of the execution when network connectivity is lost or another failure occurs. The authors of [74] propose a lightweight application partitioning mechanism to achieve seamless computational offloading. The partitioning decision

is transformed into an optimization problem and solved by the optimization solver. Various performance parameters are used to fix application-based partitioning of Android applications. Android services use Inter-Process Communication channels to perform Remote Procedure Call (RPC), the middleware intercepts the requests sent to the services and decide whether the request will be sent to local or cloud services. It reduces local execution time by considering multiple environment variables in the cost function of the optimization problem including data transfer cost, CPU cost, and memory cost of the mobile device. The constraints are related to minimization of memory usage, energy usage, and execution time. A formal description and modelling of these variables will be presented in Chapter 4.

2.2.2 Multisite Offloading

There are different ways to classify current mobile computation offloading frameworks in the literature. Existing surveys and analysis studies are either based on mobile cloud application models and offloading objectives [143], challenges in designing future mobile cloud applications [150], application partitioning (how the tasks are identified for offloading) [128] [85], or context-awareness [103]. We use cloud resource types as our classification methodology by focusing on infrastructure destinations for offloading. More precisely, answering the question of "where to offload" in mobile cloud offloading aspects [152].

There are numerous mobile computation offloading frameworks and mechanisms in the literature that aspire offloading of mobile tasks to a single or multiple servers in the same architecture tier. We consider this type of offloading to be a “Single Site Offloading” mechanism. The offloading process could be between the host mobile device and datacenter cloud servers over WAN communications through the Internet. Other works introduced Cloudlets in the LAN of the mobile device to solve the high latency issues associated with offloading to distant cloud servers. Moreover, some researchers considered using mobile devices as both service consumers and service providers in the formation of MACs.

The solutions that address offloading mobile applications to Remote Clouds use an infrastructure-based approach which refers to public cloud services including IaaS, PaaS, and SaaS. The mobile device offloads its computation to public cloud services via WiFi or cellular networks. Some solutions use an image of the mobile device operating system hosted in the cloud servers, forming a one-to-one relationship, such as in CloneCloud [31]. Other approaches of remote cloud-based offloading include MAUI [35], ThinkAir [73], Odessa [108], COSMOS [123], and Cuckoo [69].

As per the description in Section 2.1.1.1, cloudlets are trusted and resourceful computers deployed as middle layer servers between mobile devices and remote cloud servers to reduce latency and improve the mobile application performance. It was originally introduced by Satyanarayanan et al. [120] followed by enormous works such as [147] [48] [26].

Nearby Mobile Devices can form an ad hoc cloud among themselves using either Mobile Ad-hoc NETwork (MANET) through short-range wireless networks such as WiFi-direct and Bluetooth in dynamic topologies, with no support of networking infrastructure or infrastructural WiFi using Zero Configuration Network technologies [30]. Nearby local mobile devices can provide an even lower latency for computation offloading in case cloudlets or public cloud servers are out of reach of the Host Mobile Device. The most notable works in this category include Hyrax [90], Scavenger [75], Serendipity [124], and Cirrus [122].

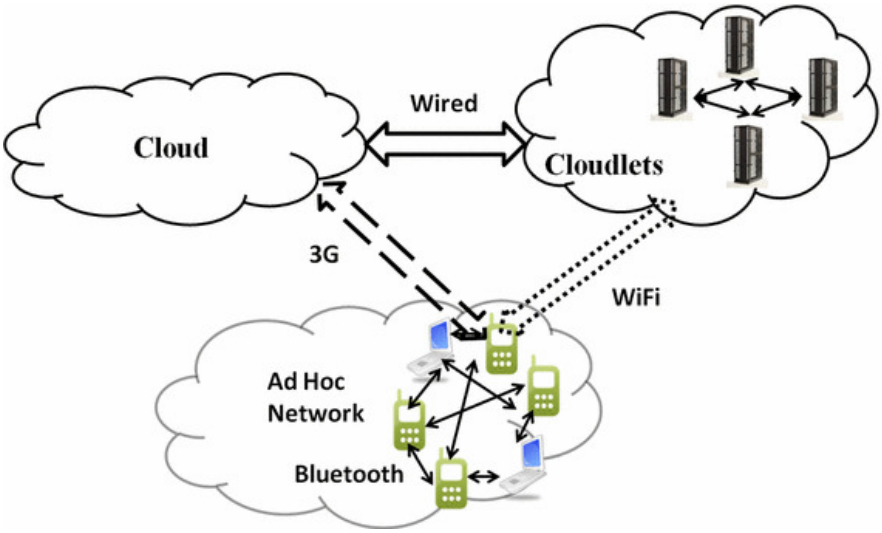


Fig. 2.4 A Proposed Multisite Offloading System Architecture [112]

With the increase in the number of heterogeneous devices all around us, future computation offloading architectures with multi-tiers will become relevant. As an example, we can contemplate a smartwatch which is connected to a smartphone via Bluetooth. The smartphone can be connected to a cloudlet in the vicinity via WiFi or to an edge server in the mobile operator base stations using cellular networks. The cloudlet or the edge server are also connected to a cloud datacenter via a fixed network. In such an n-tier system, the heterogeneous devices differ significantly in terms of energy, computation, and communication resources, increasing typically from the very limited (tier 1: smartwatch) to the virtually unlimited (tier 4: cloud servers).

In the next chapter, existing papers, publications, and frameworks in multisite offloading are presented and reviewed and a taxonomy of the primary studies is constructed.

2.3 Tools and Technologies

This section describes some of the background technologies that are used throughout this thesis and in the work of producing MAMoC. It starts with the two important tools that are used for designing a lightweight runtime environment for the components running on MAMoC Server. These choices differentiate MAMoC from most of the works in the literature described in the previous chapter. Zero Configuration Network and Wi-Fi P2P, which are discussed in Subsection 2.3.3 and Subsection 2.3.4 respectively, are important communication mechanisms used in the offloading task transfer between the Nearby Mobile Devices. Web Application Messaging Protocol discussed in Subsection 2.3.5 and its two subsequent technologies are used for device to server communications in MAMoC. These technologies will be revisited in Chapter 5 with their design and implementation references.

2.3.1 Containers

Containers are utilised to host the server components of MAMoC. Linux Containers (LXC) is a virtualization method for running multiple isolated Linux systems on a single machine. Docker[1] extends LXC to automate the deployment of applications inside software containers.

For our published work in [136], we used an already developed Swift Docker image to implement a server-side Swift application to accept incoming requests from mobile devices. The developed container provides an environment ready to be customised for other mobile applications. It provides a feature-rich yet lightweight execution environment for offloaded tasks. In the subsequent works of redesigning MAMoC Server into a new environment, the choice of using containers persisted for the server-side modules.

[1] https://www.docker.com/

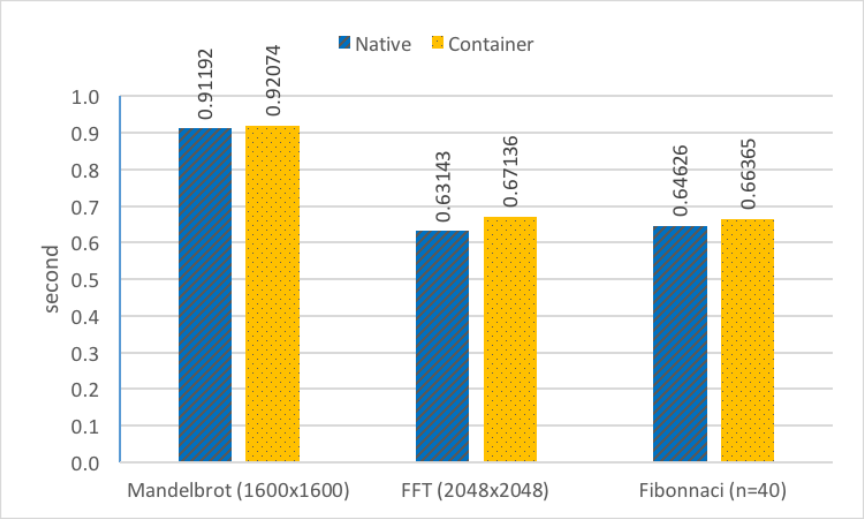


Fig. 2.5 Container vs. native benchmarking

To test the performance of containers, we decided to run three types of compute-intensive workloads to experiment with the speed of computation on Docker containers and on native platforms. As the results show in 2.5, the computation speed of containers is close to native platform performance within 1% region [43] [145]. To ease the creation of the container, we provide a Dockerfile that can automate the process of creating a container for the server-side part of MAMoC. The container is pre-configured with the necessary build environment for handling the client requests.

Despite the ease of deployment and performance boosts, using Docker for deploying offloading service providers has the following benefits:

- It provides the edge and cloud app developers with the choice of programming languages and framework. The library that MAMoC uses in designing the server components provides libraries for most popular programming languages. MAMoC Server is written in Python. However, the server logic can be rewritten by any developer wishing to use Javascript or other languages.

- Enables efficient, secure, and automated distribution of application updates to edge devices. The deployment configuration of *ImageAlwaysPull allows our containers to always host the newest images that are pushed to the Docker hub registry.*

2.3.2 Android-x86

Android-x86 is an open source project that deploys an Android operating system to an x86 or AMD hosts [7]. Many previous studies such as [22][31] have used it to deploy a virtual mobile environment. This is not chosen as MAMoC's server-side solution for the following reasons:

1. It needs a VM with an updated Android-x86 image. The authors in [46] discuss the extra overhead that comes with full virtualization of Android-x86 in the server which is shown to consume substantial CPU resources and to slow down the performance of code execution [142].

2. As it was shown in Section 1.1, the size of the image is large even when it is deployed on containers [155].

3. It is difficult to keep the x86 versions up to date with the new Android versions. With new Android versions releasing every year, it is difficult for Android x-86 to catch up with all the new updates.

4. The decompilation and reverse engineering tools for Android have advanced in the last few years. There are many well-researched academic and powerful industrial tools that aid with the process of decompiling Android PacKage file (APK)s.

5. The modern mobile application development programming languages such as Kotlin and Swift have server-side execution in mind. Both Kotlin [2] and Swift [3] support server-side executions. There are even some active open-source server-side frameworks such as Kitura and Vapor. Vapor was used in one of our previously published work [136].

Most of the computation offloading frameworks which are implemented for Android applications utilise ThinkAir [73] offloading model which requires an instance of Android x-86 in the server. In this scenario, the server unpacks the offloaded code and uses Java Reflect [4] to create a new instance of the offloaded class. It will then invoke the annotated method to execute it with the parameters and return the results to Host Mobile Device.

[2]https://kotlinlang.org/docs/reference/server-overview.html
[3]https://swift.org/server/
[4]https://developer.android.com/reference/java/lang/reflect/package-summary

2.3.3 Zero Configuration Network

Service discovery lets devices spontaneously become aware of the availability and capability of peers on the network, so clients can discover and consume services without prior knowledge of them. The goal of the service discovery protocol is to decrease or eliminate explicit administration [71]. The Zero Configuration Network (ZeroConf) [30] is a service discovery mechanism which enables seamless discovery and interaction of devices and services without prior configuration. It is an unstructured and decentralised discovery protocol that offers service discovery in local and ad-hoc networks. It supports the advertising and discovery of services using link local addressing and multicast DNS (mDNS). Service publication is performed by multicasting the service advertisement to all the network devices. The devices do not need a pre-configured network to exchange data with each other. There is no configuration needed because they can discover each other via multicast DNS (mDNS) [28]. mDNS is a service that resolves hostnames on a local network without the use of a central domain name server. After the connection is established between two devices, data can be exchanged between them in either reliable mode over TCP or unreliable mode over UDP. The reliable mode guarantees the delivery and avoids out-of-order packages, so this mode is used throughout framework data transfers. Instead, a resolving host simply sends a DNS query to a local multicast address and the host with that name responds with a multicast message with its IP address.

Multicast DNS is also used in combination with DNS based service discovery (DNS-SD) [29] where a host that provides a network service can announce its service to LAN. Those services can then be discovered using multicast messages. Each mobile device can advertise services and discover what services are offered by other devices in the proximity. A browser object in the Host Mobile Device searches for peers that have an advertiser object. In this thesis, DNS-SD protocol has been implemented to be aware of the availability of local machines. The network resources are classified according to the DNS-SD naming structure. DNS SRV and DNS TXT records are used to facilitate this protocol [29]. A DNS query in a specific format is sent to a local network in order to find available services. The format of a service type which is specified as “<Service>.<Domain>” make all service instances available in that domain.

2.3.4 Wi-Fi P2P

Wi-Fi P2P is a direct communication technology between terminals with no intermediate devices such as routers and Access Point (AP)s. Wi-Fi module is

basically mounted on most smart devices. The 1.7 version of the Wi-Fi P2P technical specification [151] was released in 2016. Wi-Fi Direct enables devices to discover each other and form a P2P group. In each P2P group, one elected node, called Group Owner (GO), functions as an AP. This standard enables devices to connect with each other without requiring a wireless AP (a.k.a router or Wi-Fi hotspot). After a widespread of the Wi-Fi technology, the capability of P2P device connections has been extended to it versus the conventional approach of using APs. Specifically, the Wi-Fi Direct technology is developed by the Wi-Fi Alliance to broaden the use cases of Wi-Fi technology for D2D communications [19] also called WiFiP2P technology [37]. Other D2D connectivity technologies such as Bluetooth has nominal ranges from 10 to 100 meters and transfer speed between 250 Kbps to 25 Mbps [79]. Wi-Fi Direct inherits all the capabilities from IEEE 802.11 standards and provides a nominal range up to 200 meters and transfer speed up to 250Mbps. Power saving support and extended QoS capabilities; Wi-Fi Direct can be considered one of the most promising candidates for a wide range D2D communication and suitable for the goal of distribution across mobile devices.

Wi-Fi P2P uses mobile devices as AP for connecting with other devices without the need for a network infrastructure. It essentially allows the creation of software APs by extending the existing AP and Station (STA) client-server architecture with introducing a GO and a Group Client (GC) that can connect in an ad hoc or P2P mode. A GO is an "AP-like" entity that can set up multiple P2P links with GCs. A device can be optionally configured to operate simultaneously as a AP and as a legacy device in a Wi-Fi network for the need for concurrent connections. A GC is a Wi-Fi P2P-compliant device that may connect to a GO. Besides the Group Owner/Client functions, Wi-Fi P2P also specifies a power-saving feature to make it suitable for battery-powered devices, and *Wi-Fi Protected Access 2 (WPA2)* has been implemented to provide security protection.

This group formation starts from scanning in the discovery phase. When an existing P2P group is found, the device joins the ad hoc cloud. If no group is encountered, the device can announce itself as GO and let other mobile devices connect to it. When devices discover each other, they may enter a negotiation phase to ascertain which device would be the GO and serve as a AP. Upon a successful connection with the GO, the GO sends information about all the other devices are in MAC. The new device can then connect to the other MNs in the network as required. Using Wi-Fi direct can reduce the network level complexity, such as discovering new devices in the network, connecting these devices and even notifying the connected devices when any particular device drops from the network.

2.3.5 Web Application Messaging Protocol

As the demand for multisite mobile clouds rise, service providers, enterprises, regional cloud providers will need to set up next-generation mini and micro servers across multiple locations to satisfy modern mobile application requirements. The operational complexity increases with this new architecture and there will be a need for a counterbalance and a highly automated network communications between the mobile devices and servers that can make multiple edge and public cloud nodes appear as one logical entity to streamline the management of multiple offloading sites.

Web Application Messaging Protocol (WAMP) is an attempt at developing an open, text-based standard that combines Publish/Subscribe with Request/Response patterns, complex routing and delivery strategies [101]. It is most commonly used in conjunction with the Crossbar[5] router and Autobahn[6] client libraries. There are a plethora of protocols that are adopted for message communications. A comparison table with other protocols such as MQTT, AMQP, and XMPP. is presented in their website [7]

2.3.5.1 Remote Procedure Call

RPC systems provide a request-response model similar to local procedure calls. It is a powerful technique for designing distributed, client-server based applications. When an application performs a RPC, the system constructs a request message that identifies the procedure to call and contains serialized representations of any arguments. Calls are performed on static code so the environment for each call is constructed at call-time from the passed arguments and any static data referenced from the code. After the call is completed, a response message containing serialized representations of the return values is sent back to the caller. Instead of accessing remote services by sending and receiving messages, a client invokes services by making a local procedure call. The local procedure hides the details of network communication. RPC provides a different paradigm for accessing network services.

RPC messaging pattern involving peers of three roles: a *caller* that issues calls to remote procedures by providing the procedure URI and any arguments for the call. The *callee* will execute the procedure using the supplied arguments to the call and return the result of the call to the *caller*. *Callees* register procedures they provide with *dealers*. *Callers* initiate procedure calls first to *dealers*. *Dealers*

[5]https://crossbar.io
[6]https://crossbar.io/autobahn/
[7]https://wamp-proto.org/comparison.html

route calls incoming from *callers* to *callees* implementing the procedure called, and route call results back from *callees* to *callers*. The *caller* and *callee* usually run application code, while the *dealer* works as a generic router for remote procedure calls decoupling *callers* and *callees*. In MAMoC, SNs are the callers while the ENs and PNs are callees. MAMoC router component in MAMoC Server serves as the dealer between the two sides.

2.3.5.2 Publish/Subscribe

Publish/Subscribe (Pub/Sub) is a messaging pattern involving nodes of three aspects: *publishers* publish events to topics by presenting the topic URI and any payload for the event. *Subscribers* of the topic will receive the event jointly with the event payload. *Subscribers* subscribe to topics they are concerned in with Brokers. *Brokers* route events incoming from publishers to subscribers that are subscribed to various topics. The *publisher* and *subscriber* normally run application code, while the Broker serves as a generic router for events decoupling *publishers* from *subscribers* [139].

2.4 Summary

In this chapter, an analysis of the existing mobile cloud architectures is given including the three models of MCC and the more current trend towards the allocation of computation near the user in the form of Edge/Fog Computing paradigms. It is also shown that while these paradigms share common grounds, they also have many differences. The main difference between a more traditional mobile cloud is in the distance to the final client. While MCC is focused on enabling computation offloading capabilities to mobile devices, EC and FC focus on providing a service closer to the client, thus reducing latency between the client and the public cloud instances in the datacenter. A number of technical tools including containers and Android-x86 and communication mechanisms were then discussed to enable readers understand the design choices taken in the subsequent chapters for designing MAMoC.

Chapter 3

Literature Review

3.1 Overview

This chapter aims to develop a survey and taxonomy of existing research in multisite MCO studies in MCC research area. As described in Chapter 2, multisite offloading approaches pick one or multiple surrogates from a set of candidate sites (here we use "node" and "site" interchangeably), i.e., there could be multiple offloading destinations, where both cloud VMs and mobile nodes can serve as service providers. In some studies, this approach is also called Hybrid Offloading [2], Heterogeneous Mobile Cloud Computing [164], Multi-tier MCC [153], or 2-tier mobile cloud architectures [109] [110] [157]. Our study only focuses on offloading schemes where multiple surrogates (i.e. servers, destinations, sites) are available for service providers for Host Mobile Devices.

On one hand, the consideration of multiple surrogates results in a more flexible offloading model for running mobile cloud applications. On the other hand, new challenges arise when building a multisite computation offloading system because of the dynamic nature of the environments they operate in. The connection to an offloading site might not be available when needed or might become unavailable during the offloading operation. Moreover, the offloading sites have different processing capabilities, thus increasing the level of uncertainty in reaching the desired qualities of the mobile cloud applications.

First, Section 3.2 discusses the search methodology in collecting the data for conducting the review and constructing the taxonomy based on the methodology guidelines proposed by Kitchenham et al. [18]. Section 3.3 presents a systematic literature review where we derive a taxonomy of the current multisite computation offloading solutions. Since the mobile cloud offloading topic involves many entities, the best effort has been made to cover the significant works in the two aspects of

"adaptive" and "multisite" offloading and the corresponding techniques used in this research area. Finally, Section 3.4 presents a classification based on these works according to the supported properties to find the current trends and evaluate them to suggest pertinent directions for future mobile cloud offloading systems and how MAMoC fits within the discovered gaps.

3.2 Survey Methodology

The goal of this survey is to identify work in mobile cloud offloading systems with a multisite computation offloading perspective. To achieve this goal, we define the following research question:

Which solutions and frameworks in MCC studies identify multisite computation offloading in the literature?

The three main keywords derived from the research question are: *multisite*, *offloading*, and *mobile cloud*. Each of these keywords has a set of related synonyms and alternative spellings. Based on these keywords and their related terms, the following basic search string was defined:

("multisite" OR "multi-site" OR "three-tier" OR "multi-tier" OR "multiple servers" OR "multiple surrogates" OR "multiple nodes") AND ("computation offloading" OR "code offloading") AND ("mobile cloud" OR "mobile edge cloud" OR "cloudlet")

We performed the wildcard search in the most known scientific databases including ACM Digital Library, IEEE Explore, Scopus, Science Direct and SpringerLink. The number of returned computing and research article results from each digital library is shown in Table 3.1.

Table 3.1 Digital Library Database Search Results

Search database	**URL**	**Results**	**Included**
ACM DL	dl.acm.org	8	3
IEEE Xplore	ieeexplore.ieee.org	10	4
Scopus	www.scopus.com	28	2
ScienceDirect	www.sciencedirect.com	44	3
Springer Link	link.springer.com	54	5

There were multiple duplicate results from a total of **144** fetched items. Through duplication-checking tool provided by Mendeley [1], the number of results was reduced to **82** papers.

Besides this technique, a handful of papers that were discovered in the course of my PhD are selected manually [2]. We then started excluding the surveys and analysis studies. **35** papers were found to be either survey studies addressing a variety of issues, challenges, and opportunities of mobile cloud offloading systems or studies that were a subset of a larger study by the same author(s). We then started evaluating the remaining papers against the inclusion/exclusion criteria based on the title, abstract, keywords, and an initial scan of the paper.

The following inclusion/exclusion criteria were reviewed during the process to ensure that the results represented multisite computation offloading in mobile cloud studies:

- Studies in which the Host Mobile Device is augmenting its computing power by using multiple surrogates such as nearby devices or far cloud resources. The offloading solutions where there are more than two tiers involved in the offloading architecture are considered. Even though some studies such as [109], [110], and [157] call it two-tier cloud architecture, the mobile device is not considered as a tier. Therefore, those studies are still perceived as mobile-cloudlet-cloud 3-tier architectures.

- The offloaded task execution must be performed by more than one service provider. The more traditional single server offloading studies such as CloneCloud [31] and MAUI [35] are not included in our survey.

- The single-tier offloading studies are also included. Even though studies that investigate offloading to local mobile device clouds can offload computation to multiple Nearby Mobile Devices, they are still in the same architecture tier.

- The multi-user offloading studies are also excluded. An example of such works is Cardellini et al. [20] which considers both nearby Cloudlets and Remote Cloud servers in modelling the competition of mobile devices on shared resources of the hybrid cloud as a game where each Host Mobile Device decides which subtasks to be offloaded and to which local server or the cloud. We only investigate the offloading studies where the focus is on a single mobile user with multiple tasks and multiple offloading destinations.

[1] https://www.mendeley.com
[2] https://www.connectedpapers.com/

- The studies where the servers are managed by the network operators in the coordination of resource exchange between mobile users are not considered. Many recent studies in the field of MEC have focused on this type of study [1]. We limit our study to multisite offloading within MCC research area.

We further excluded **29** papers due to their irrelevance based on our employed inclusion/exclusion criteria. The final number of considered papers are **20** (17 from the wildcard search results and 3 manually included by us) primary studies.

3.3 Taxonomy

This section identifies common themes, characteristics, requirements and challenges based on the multisite offloading publications collected in the literature. The taxonomy sections and their types are displayed in Figure 3.1. The Table 3.2 lists all the collected works and categorises them using our proposed taxonomy.

3.3.1 Offloading Objectives

As described earlier in Section 2.2 that offloading is performed to achieve a number of goals. The two most popular offloading objectives in the literature are to improve the application performance by decreasing the execution time of the tasks (makespan) and to reduce the overall energy consumption to prolong the battery life of the constrained mobile devices. We found in the primary studies that some works perform multisite offloading to also increase the service availability for the mobile devices.

To decrease the overall response time of face detection and recognition application, MOCHA [133] transfers the captured images from mobile devices to nearby Cloudlets through a high bandwidth and low latency WiFi connection. The cloudlet can dynamically partition and process the distributed tasks in a parallel fashion to the Amazon EC2 instances on the cloud using the high-speed Internet connection. Moreover, by minimizing the latency from the user to the cloud, less mobile device energy is consumed.

Most of the works [140], [161], [125] build an energy optimization model that takes into account the local processing energy consumption, the bandwidth rate between the mobile device and offloading performance difference of k different candidate sites. The goal is to partition the application under which the energy consumption of the mobile device is minimized.

The existence of multiple service providers and the tight communication between them is an important aspect of multisite offloading. This feature allows

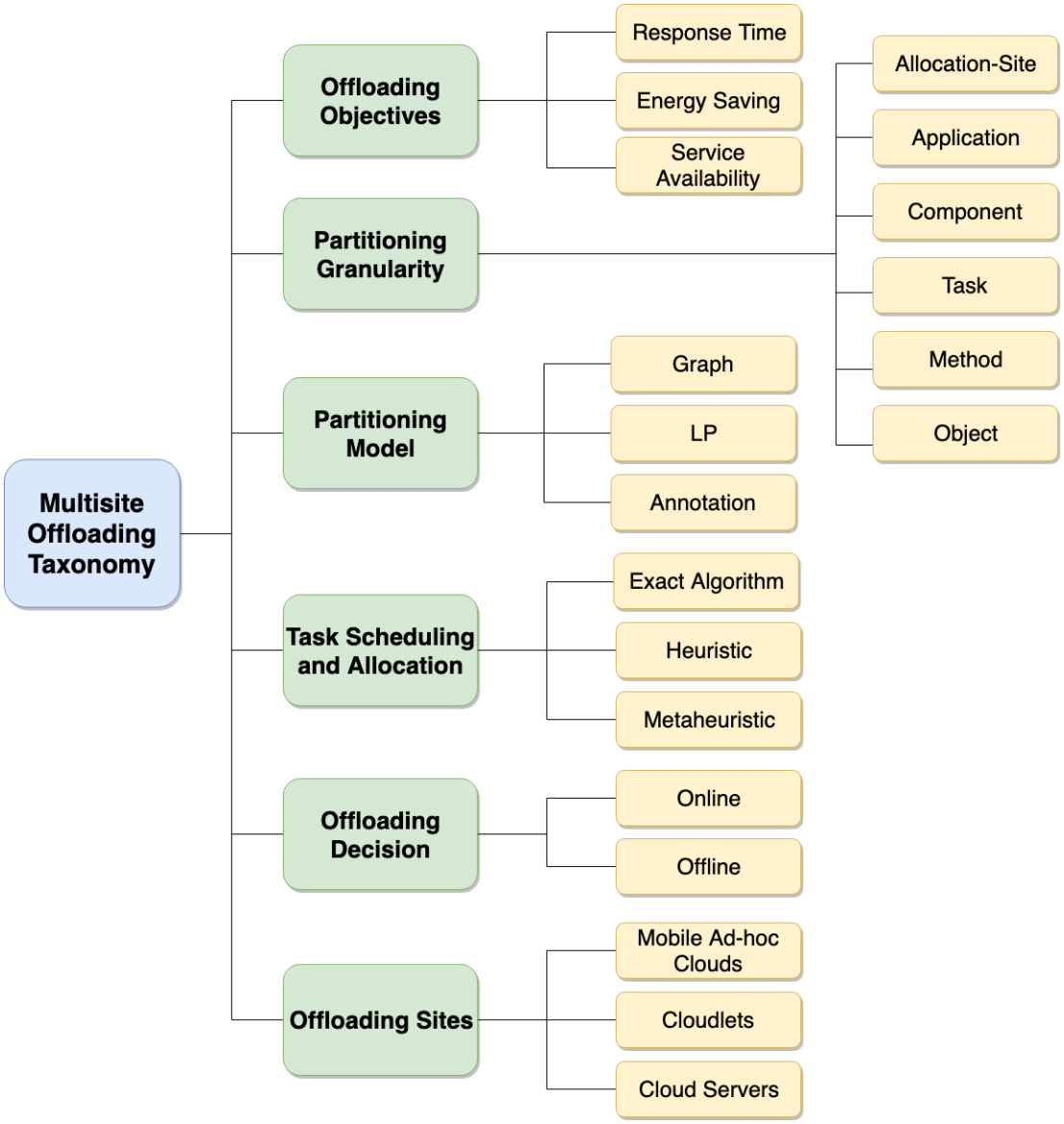


Fig. 3.1 A taxonomy of multisite offloading mechanisms

transferring services between the different devices to solve the mobility issues of mobile users, which results in increasing service availability. Some researchers have tried to increase the number of available service providers for mobile users. Due to the mobility issue of mobile users, services can be interrupted before the offloading process is over. To increase the reliability and availability of services, authors in [112] use a handoff algorithm to migrate the computation to another service provider when the mobile user is expected to move away from the current surrogate. Meanwhile, Chen et al. [23] improves the availability of services by selecting a standby service in order to ensure application performance is not degraded when the original service cannot serve it any longer.

MuSIC [110] claims to provide fair use of Cloudlets and Remote Clouds servers. However, the user access limitations on wireless APs are not considered in the work. Therefore, Xia et al. [157] improve the concept of a fair share of external servers by considering the user access limitations on APs which cannot be ignored in real-world scenarios. Similarly, MAGA [125] considers the mobility of users and multiple APs in their model when trying to increase the available services for a mobile user.

3.3.2 Partitioning Granularity

Application partitioning is the process of decomposing the mobile application into multiple dependency tasks for local and remote executions (to answer the query of "what to offload"). It is a technique of splitting up the application into fragments while retaining the semantics of the original application [85]. It has to be performed before offloading. For example, a multimedia application such as face recognition may contain several compute-intensive image analysis modules with various resource requirements. It provides the task graph to a task scheduling algorithm. The tasks will later be offloaded to their suitable resources according to the scheduling and allocation results. Among studies conducted on MCC and of different offloading middlewares that have been presented in this area, *what to offload* is the principal issue that has attracted the most attention.

MCO overcome the resource limitations of lower-end devices by splitting resource-intensive tasks and allocating subtasks to other resourceful devices. According to [4], the computation offloading frameworks can be classified into frameworks based on a virtual machine cloning and the systems based on code offloading. Code offloading can be further classified according to their offloading granularities ranging from tasks [108] and methods [73] to thread-level partitioning [31].

In our primary studies, several granularities are used ranging from coarse-grained partitioning applications to components in [153] [140] [53] [125] [161], to tasks in [133] [110] [126] [112], to finer-grained partitioning of methods in [164] [13] [23] [96] and objects in [99]. Although fine-grained remote execution increases flexibility despite having access to other offloading options, coarse-grained is employed in most of the works because of the possibility of increasing the efficiency with amortizing overhead over a larger unit of execution in a site. As a result, coarser-grained partitioning is suitable for applications that seek to reduce the transmission overhead between the different parts of the application, and therefore, not appropriate for applications with shorter operations.

Table 3.2 A taxonomy of the multisite offloading works

#	Year	Work	Objectives	PG	PM	TSA	OD	OS	EP
1	2011	Sinha et al. [131]	RT	Allocation-site	LP/Graph	Exact	Offline	Cloud	Simulation
2	2012	MOCHA [133]	RT	Task	-	Heuristic	Offline	Cloudlet / Cloud	Simulation
3	2013	MuSIC [110]	RT/SA	Task	-	Metaheuristic	Online	Cloudlet / Cloud	Simulation
4	2013	ENDA [82]	ES	Application	-	Heuristic	Online	Cloudlet / Cloud	Simulation
5	2013	EMSO [99]	RT/ES	Object	LP/Graph	Heuristic	Online	Cloud	Simulation
6	2014	Xia et al. [157]	ES/SA	Task	Graph	Heuristic	Online	Cloudlet / Cloud	Simulation
7	2014	MMRO [153]	RT/ES	Component	Graph	Metaheuristic	Offline	Cloud	Simulation
8	2014	Shih et al. [126]	RT	Task	OpenCL	Heuristic	Offline	MAC/Cloudlet/Cloud	Simulation
9	2015	Ravi et al. [112]	ES/SA	Task	Annotation	Exact	Online	MAC/Cloudlet/Cloud	Testbed
10	2015	mCloud [164]	RT/ES/SA	Method	Annotation	Exact	Online	MAC/Cloudlet/Cloud	Testbed
11	2015	Cheng et al. [27]	RT	Task	Graph	Metaheuristic	Offline	MAC / Cloud	Simulation
12	2016	EMOP [140]	ES	Component	Graph	Metaheuristic	Online	Cloud	Simulation
13	2016	Enzai et al. [41]	RT/ES/CR	Task	Graph	Heuristic	Offline	Cloud	Simulation
14	2016	CoBOS [13]	RT/ES	Method	Annotation	Exact	Online	MAC/Cloudlet/Cloud	Testbed
15	2017	FHMCO [53]	RT/ES	Component	Graph	Metaheuristic	Offline	Cloudlet / Cloud	Simulation / Testbed
16	2017	Chen at al. [23]	RT/SA	Method	-	Exact	Offline	Cloudlet / Cloud	Testbed
17	2017	Jin et al. [65]	RT	Component	Graph	Metaheuristic		Cloud	Simulation
18	2018	MAGA [125]	ES/SA	Component	-	Metaheuristic	Online	Cloudlet	Simulation
19	2018	CRMGA [161]	ES	Component	Graph	Metaheuristic	Offline	Cloudlet / Cloud	Simulation
20	2018	ULOOF [96]	RT/ES	Method	Annotation	Exact	Online	Cloudlet / Cloud	Testbed

PG: Partitioning Granularity, **PM**: Partitioning Model, **TSA**: Task Scheduling & Allocation, **OD**: Offloading Decision, **OS**: Offloading Sites, **EP**: Evaluation Platform
Objectives: **RT**: Response Time, **ES**: Energy Saving, **SA**: Service Availability, **CR**: Cost Reduction

3.3.3 Partitioning Model

The partitioning model specifies how the mobile parts are represented in the application. The application can be partitioned through a diversity of strategies. A mobile application developer can explicitly select the parts of the application that should be offloaded using special static annotations (e.g., @Offloadable [31], @Remote [73], @Remotable [35], @OffloadingCandidate [96]). Thus, once the application is installed in the device, the mechanism selects the annotated parts to be offloaded. In contrast, automatic strategies need to estimate the offloadable parts through the use of static analysis and dynamic profiling tools. Automated mechanisms are preferable over manual approaches, as they can tailor the code to be executed on different devices. Thus, automated mechanisms overcome the problem of application development, which consists of adapting the application every time it is installed in another device with different computational capabilities.

Graph-based and Linear Programming-based algorithms are common approaches for determining the application partitioning model [105]. The graph-based partitioning algorithms work on an application graph consisting of vertices and edges and the calling relationships [148]. The LP-based algorithm is applied to profile the model of the present situation and the constraint formula.

In the graph-based partitioning, the vertices of the graph can represent the states and computation costs of an application unit while the edges can represent communication costs or dependencies between the different units. The number of vertices and edges of the graph may differ depending on the granularity on which the application is modelled as outlined in the preceding section. For multisite partitioning, the graph is partitioned into two or several parts to be executed on different resources. An appropriate graph can be effective in making optimal decisions that affect the efficiency and quality of offloading. A range of approaches to graph partitioning has been introduced and implemented in the literature [148].

Xia et al [157] use a weighted bipartite graph to represent the number of location-aware mobile tasks and corresponding nodes that are computing facility allocations. Similarly, in MMRO [153], an application is represented as a graph $G = (C, E)$ where a vertex is a component and an edge is an interaction between the components. The objective is to discover a mapping from the application graph to the network $G_{sur} = (S, L)$ of offloading sites where S is a site and L represents a network connection link between two sites. The objective function is to achieve offloading some computation workload from an original mobile device and distribute them onto candidate sites.

Cheng et al. [27] generate a weighted direct task graph of wearable device applications. The task graph contains the time intervals between any two tasks that are executed on a wearable device not to be greater than a specified threshold. Similarly, EMOP [140] models mobile applications as weighted direct acyclic graphs where the vertices denote a component and the edges denote the communication channel between them. The primary focus of EMOP is in formulating multisite partitioning model to minimise energy consumption. A Markov Decision Process is used to estimate mobile wireless fading channels for offloading tasks to multiple cloud servers.

Enzai et al. [41] also represent the application as a direct acyclic graph where the vertices are 2-tuple computation tasks of data volume and the number of instructions. The intention is to identify an optimised task allocation for a service among a set of the available services.

FHMCO [53] models the applications as weighted relation graphs where weighted vertices denote the application units and their execution time on a particular site and the edges depict the invocations between them. The graph is then adopted by the decision engine to find a near-optimal partition of components to be offloaded to the offloading sites as we describe it in the next subsection.

Finally, CRMGA [161] uses a static analysis tool to generate the Object Relation Graph (ORG) for the application. It then uses dynamic profiling to construct Data Cost Graph based on the node and edge weights of the ORG.

Linear Programming-based partitioning is a mathematical approach used for finding the best amount that can maximise or minimise an objective function based on energy consumption or execution time. In this approach, the objective function is formulated as an optimisation problem and is then calculated using the technique of linear programming for the best options for achieving the objective. LP-based approaches are appreciated for their ability to find an optimized solution for a given objective function. However, solving such problems requires a long computational time [53].

Sinha et al. [131] made use of Integer Linear Programming (ILP) to solve optimization equations. Similarly, Niu et al. [99] models the application partitioning as a 0-1 Integer Linear Programming problem. Both approaches solve the optimization problem using a multi-way graph partitioning algorithm [148]. The objective of this partitioning approach is to get the best trade-off between computation costs and communication costs of an application. However, Sinha et al. [131] perform the partitioning in allocation-site-level but Niu et al. [99] construct the application as a Weighted Object Relation Graph (WORG) so the partitioning is performed at the object level to achieve more precise offloading.

By annotation-based partitioning model, we refer to programmer-defined partitioning which is performed during the development of the mobile application. The annotations can be plugged at different granularity levels such as classes and methods. Ravi et al. [112] use Java Reflection[3] to identify the offloadable methods using *@offloadable* annotation for offloading them to cloud and interactive methods using *@interactive* to offload them to Nearby Mobile Devices. Similarly, mCloud [164] uses *@OFFLOAD* annotation to annotate the methods that will be processed at compile-time and inspected at run-time to be considered for offloading according to the solver component. Finally, CoBOS [13] uses the provided *Offloadable*, *OffloadExecution*, and *OffloadIO* annotations in the core class library *java.lang* to distribute application parts from a mobile device to participating Offload Execution Engines. Application developers can then extend the abstract class *OffloadExecution* that possesses the abstract Java method *execute*, which is annotated with *Offloadable* that marks it as a suitable candidate for remote execution.

With the exception to other works, Shi et al. [126] uses Open Computing Language (OpenCL)[4] which is a parallel programming model. OpenCL abstracts the hardware details from the developers by dispatching the workloads to heterogeneous cloud resources. Therefore, an application can be executed on different processing elements when it is deployed on different hardware platforms. With OpenCL, the application developers do not need to manually specify the offloaded parts for the applications.

3.3.4 Task Scheduling & Allocation

The task scheduling algorithm is the process of allocating the tasks to the respective available resources (to answer the question of "where to offload"). It calculates the optimal resource orchestration strategies according to task requirement, resource availability, and mobile device status from the monitoring and profiling modules. The task graph from the partitioning module and profiling information are inputs into the algorithm and translated into the solution space. Some multisite works in the literature follow an exact algorithm to schedule the tasks to the respective resources while others use a heuristic or a metaheuristic to find near-optimal solutions. The employed algorithms in the multisite works are listed in Table 3.3.

MCDM algorithms are used in two of the primary studies for selecting offloading destinations according to different criteria and user preferences. Ravi et al. [112]

[3]https://docs.oracle.com/javase/tutorial/reflect/index.html
[4]https://www.khronos.org/opencl/

Table 3.3 Task scheduling and allocation algorithms used in our primary studies

Work	Exact	Heuristic		Metaheuristic				
		GS	HC	GA	AC	PSO	SA	SO
MOCHA [133]		✔						
MuSIC [110]							✔	
ENDA [82]		✔						
EMSO [99]	✔							
Xia et al. [157]		✔						
MMRO [153]					✔			
Shih et al. [126]		✔						
Ravi et al. [112]	✔							
mCloud [164]	✔							
Cheng et al. [27]				✔				
EMOP [140]								✔
Enzai et al. [41]			✔					
FHMCO [53]		✔				✔		
Chen et al. [23]	✔							
Jin et al. [65]				✔				
MAGA [125]				✔				
CRMGA [161]			✔	✔				

GS: Greedy Search, **HC**: Hill Climbing, **GA**: Genetic Algorithm, **AC**: Ant Colony, **PSO**: Particle Swarm Optimization, **SA**: Simulated Annealing, **SO**: Stochastic Optimization

uses MCDM to choose the best possible resource to offload the computation tasks while using a handoff strategy for offloading tasks to different resources. Similarly, mCloud [164] uses multiple criteria of the wireless mediums such as the energy cost of the channel, the link speed, etc. to select the best interface under current context such as data rate, workload size to obtain the best data transfer performance and minimise energy consumption. It is worth noting that mCloud [164] also uses a Min-Min algorithm to select the resource with the minimum response time and energy consumption for offloading the tasks to Nearby Mobile Devices.

The scheduling problem to multiple providers is a typical constrained optimization problem. Because of the NP-completeness nature of assigning multiple tasks to multiple offloading sites, most of the studies consider a heuristic or a meta-heuristic for finding a "good enough" solution instead of an optimal one for the scheduling problem. Heuristics allow for solving scheduling problems faster than traditional and more formal closed-form models. Heuristics can normally lead to approximate solutions close to the optimal solution of a problem [110]. For

searching algorithms, the heuristics rank the different alternatives and greedily perform a fundamental element when defining the convergence of the search. This is in contrast to the exhaustive searching [125] which is guaranteed to generate an optimal solution but could take a very large computation time when the problem size increases i.e., adding more application components and more candidates sites.

Xia et al. [157] use a greedy heuristic to determine whether the task be executed on the Host Mobile Device, or offloaded to the cloudlet or the remote cloud to minimize the residual energy ratio. The algorithm takes a set of wireless access points, a local cloudlet, and a remote cloud as an input. Having the weighted bipartite graph that we described in the previous subsection, it finds the maximum matching in the graph such that the weighted sum of all edges is minimized before offloading the tasks to different computing facilities. The time complexity of the proposed algorithm is $O((n + n_a)^3)$ where n is the maximum number of mobile users at any time slot, and n_a is the maximum number of channels at each access point.

Similarly, Shih et al. [126] use a 2-step greedy algorithm called Cost-Performance First (CPF) scheduling algorithm. The CP ratio is calculated according to the mobility state of a device (connected or disconnected). The average time complexity of the scheduling algorithm is $O(NlogN)$ dominated by the second step. The first step for finding the subtask with maximum CP according to the constraints has a linear time complexity and the second step of recursively scheduling the subtasks to the processors has a complexity of $O(NlogN)$.

The greedy search algorithm is also used in ENDA [82] to select the most energy-efficient network for offloading the application based on user track prediction, server loads and the network quality of the WiFi connection between the Host Mobile Device and Cloudlets or the cellular data connection with the Remote Clouds. Additionally, the algorithm also filters out the WiFi APs that cannot maintain connectivity along the predicted route of the user to avoid disconnection.

Enzai et al. [41] use a greedy Hill Climbing heuristic for solving a multi-objective combinatorial optimization problem. The heuristic develops an initial solution for the assignment of the tasks to the sites then greedily selects the optimal solution with the maximum target function value. The quality of the optimization results highly depends on the initial solution.

To tackle the issues of larger mobile application sizes and reduce the search space, many research works propose a metaheuristic, which is a high-level algorithm to select and guide a heuristic to generate near-optimal solutions. Metaheuristics usually adapt randomization and stochastic processes in their evolutionary searches to reduce the search space and generate solutions in a reasonable amount of time.

Cheng et al. [27] use a heuristic based on genetic algorithms. Given the weighted direct application graph, the algorithm decides how to offload each graph node among multiple services to approximate the optimal solution. MAGA [125] uses an improved genetic algorithm based algorithm that predicts the mobility of users for bringing resources closer to them. It enhances the efficiency of genetic algorithm reproduction operations by using integer encoding to envisage multiple mobile application components offloaded to multiple cloudlets scenarios. The offloading failure rate is also reduced with the mobility prediction model, resulting in higher service availability and lower energy consumption. Jin et al [65] uses a memory-based immigrants adaptive genetic algorithm for decision making of multisite computation offloading in dynamic mobile cloud environments, considering environmental changes.

MMRO [153] uses an ant colony based algorithm to calculate the offloading decision from application components to offloading sites according to a number of benefits (execution time and energy consumption) and risks (privacy and trust). The authors employ a fuzzy inference to aggregate the overall offloading benefits and compromises based on the importance levels provided by user preferences.

Rahimi et al. [110] proposes a simulated annealing based heuristic called MuSIC (Mobility-Aware Service AllocatIon on Cloud) by extending their preceding work MAPCloud [109] to support user mobility. They incorporate the capabilities of nearby local and remote cloud servers for offloading requests. The service with minimum performance based on cost, power and delay is selected as the optimal solution.

A multisite algorithm based on particle swarm optimization is proposed in FHMCO [53] which evaluates the near-optimal partitioning solution for large-scale applications. Before that, they present an optimal partitioning solution for smaller-scale applications based on an optimized Branch-and-Bound algorithm. The application cost is the sum of the cost of each application unit represented by the weighted object relation graph. While the energy cost is computed from the time cost of the energy consumption rate for the CPU and the network interface of the mobile device.

Finally, EMOP [140] uses a stochastic optimization approach by modelling the mobile wireless channels as Discrete Time Markov Chain (DTMC) and solving them with Value Iteration Algorithm (VIA) to determine the optimal offloading decision for energy-efficient multisite application execution. The algorithm finds the efficient offloading decision with a computational complexity of $O(SA)$, where S is the state space and A is the action set.

3.3.5 Offloading Decision

The decision of allocating the partitioning tasks to the sites can be performed during development or at runtime. Offline allocation decision is specified in some configuration files or in the form of annotations within the mobile application. Conversely, online allocation decision is postponed to runtime according to the load of sites and available network bandwidth among other dynamic variables. The main difference between both offline and online offloading algorithms is that offline algorithm assumes that the future predictions of the device load and network bandwidth are given, while the online algorithm considers the current progress of the application execution and the mobile device load and network bandwidth in the form of context-aware and adaptive offloading.

Sinha et al. [131] claim that by performing static application partitioning and making offloading decisions at compile-time, the overhead of dynamic profiling can be reduced. However, without dynamic partitioning, the offloading decisions cannot adapt to the dynamic changes such as connectivity failure and bandwidth fluctuation and adjust to runtime conditions since it does not take into consideration the runtime parameters such as latency and available bandwidth between the mobile device and the offloading sites. This results in unrealistic solutions in real-world mobile offloading scenarios.

Most of the works use dynamic partitioning and online offloading algorithms to accommodate runtime changes in the mobile environment. Mobile applications can be partitioned at runtime so that computationally intensive components can be handled through adaptive offloading. Many approaches use dynamic profilers to check the mobile contextual changes so that the offloading decision making is taken accordingly.

Some adaptation techniques used by the multisite works handle network variations. MOCHA [133] and EMSO [99] monitor the bandwidth changes at runtime to dynamically partition the mobile application. Most offloading systems do not explicitly select the network interface for offloading the tasks. Instead, the interface with the strongest signal or highest bandwidth is selected by the mobile operating system [15]. Although, mCloud [164] considers the different network interfaces and selects the optimal interface using the MCDM algorithm.

The mobility of mobile users causes connection losses, which result in offloading failures. Some multisite works involve predicting the user's mobility patterns. This can then be used to decide the offloading site to which the application parts should be offloaded. MuSIC [110] showed that if the mobility pattern of mobile users is known in advance, it is possible to optimally decide the best server platform. Ravi

et al. [112] use a handoff algorithm based on user mobility to migrate services horizontally between the resources in the same tier and vertically between the resources in different tiers. MAGA [125] uses an algorithm called the sequence matching algorithm, which identifies a tail matching subsequence based on a user's historical access sequence. This prediction is used for the reliability estimation of cloudlets, which is used as an important factor in the online offloading decision.

3.3.6 Offloading Sites

Once it is decided that a task or a subtask should be executed remotely, the offloading sites are selected based on the output of the task allocation algorithm. The tasks can either be sent to Remote Cloud servers, or to Cloudlets and Remote Cloud servers or to all the available cloud resources including Nearby Mobile Devices, Cloudlets, and Remote Cloud servers as we described earlier in Subsection 2.2.2.

The multisite partitioning problem and scheduling algorithms take into consideration the resource availability and computational power of the different offloading sites. The studies that focus on reducing latency between the Host Mobile Device and Remote Cloud servers such as MOCHA [133], MuSIC [110], and [125] include cloudlet-based offloading in their model.

With the availability of all the cloud resource types, some studies such as Ravi et al. [112], mCloud [164], and CoBOS [13] examine offloading to MAC resources including nearby mobile devices and cloudlet servers as well as remote cloud servers.

The proposed solutions can be evaluated using either real-world mobile applications in using physical devices in an evaluation testbed or simulation programs that emulate the real-world scenario. Several mobile applications are being developed by the studies for evaluation purposes. MOCHA [133] uses face detection and recognition application to offload them to cloudlets and remote clouds. Ravi et al. [112] use a number of different mobile applications ranging from compute-intensive, interactive, and data-intensive applications which are NQueens, an interactive game, and an image-to-text app, respectively. Similarly, CoBOS [13] uses a face recognition and a text-to-voice app. While Chen et al. [23] use a chess game as a compute-intensive app and FaceFinder as a data-intensive app to evaluate their framework.

The rest of the studies use a simulation methodology to evaluate their proposed algorithms. Even though the simulation technique is an efficient way to evaluate several performance parameters, it does not guarantee that systems behave

correctly in all possible situations especially when multiple service providers are involved in the mobile dynamic environments.

3.4 Discussion

Realising the vision of multisite offloading approach in the MCC paradigm is a challenging task because of the complexity in handling the multiple aspects involved, especially those concerned with performance evaluation, application partitioning, and tasks scheduling and allocation. After presenting the above research work from different perspectives and constructing a taxonomy for them, we present some analysis and discussion on trends in designing future multisite computation offloading systems in Subsection 3.4.1. Subsection 3.4.2 then investigates how the work in this thesis fits within the gaps found in the literature.

3.4.1 Current Trends

The following trends are identified after studying the primary works in the field of multisite MCO in particular and in the MCC research area in general.

Inter-operability & Transparency

The heterogeneity in software, hardware, and technology variations of the devices and the distinctness of computing architectures between mobile and static devices make it difficult for them to easily communicate with each other. The offloading can be done between the different systems with various computation capabilities. The Host Mobile Devices cannot transparently translate the mobile-based task sets into runnable executions directly for the static offloading resources. This lack of interpreter standard in computation offloading necessitates the developers either to manually partition the tasks into mobile execution tasks and offloading executions or to deploy an intermediate task partitioner that can translate the code across the platforms. On one hand, the manual approach requires a vast amount of work when offloading traditional mobile tasks to the server runtime environment. On the other hand, the automatic partitioning method inevitably introduces runtime computing and energy overhead.

Service Discovery

Discovering the nearby and remote cloud resources are one of the many challenges that face MCO solutions in general. None of the multisite

works that we collected propose solid discovery mechanisms to manage the offloading to multiple offloading sites. The connection between the Host Mobile Device and the k sites are assumed to be already established. Even though most of the works concede the bandwidth changes and latency issues in their partitioning and task scheduling algorithms, in real-world applications the service discovery is an essential module to monitor those dynamic changes.

Security & Privacy

The outsourcing of computation and data to external resources remains a major security concern. With the enforcement of new policies such as European General Data Protection Regulation (GDPR) [149], data security and privacy issues have become more important for cloud providers. There are risks of lacking control over data and potential data losses. Since multisite offloading involves multiple service providers, the risks of adding untrusted services and malicious nodes increase. Moreover, the orchestration of resources in the service providers might be disturbed and the reliability and dependability of the provided services can be compromised. In this case, the initialization and delivering of offloading responses may not accurately match application requirements. Standard authentication schemes and trust establishments need to be studied between mobile devices and multiple service providers.

Large-scale deployment and testing

The current multisite offloading works are either using simulation to test their solutions or develop arbitrary mobile applications and test them on a small scale in the laboratory. Given this situation, there is a need to have a larger scale of testing for globally optimized multisite offloading scenarios, considering possibly conflicting optimization parameters. This also calls for more efforts for the research of lightweight but powerful decision mechanisms to achieve this type of deployment. Specifically, supporting the legacy applications where the source code is hard to obtain or not available. Even if the code is available, understanding it can be a tedious task. Also, due to the rapid advancements in the mobile device hardware, it is hard to ask developers to update their manual annotations on the offloadable components. Automating the process of rapidly identifying the relevant parts of an unfamiliar code can greatly help future mobile applications in remote execution frameworks.

Adaptive and intelligent decision algorithms

Different computing services provide different privacy guarantees, I/O performance, computational power, available memory, and network latency. Besides, the parts of an application might have different privacy/security demands, I/O needs, CPU requirements, or latency expectations. To the best of our knowledge, there is no comprehensive treatment of these problems. Since most of the multisite offloading optimization models attempt to solve an NP-hard problem, the approximation solutions with higher performance and lower complexity are designed and evaluated in many of the studies. These heuristics can be further optimized with the use of machine learning and the latest developments in the Artificial Intelligence field. Decision making can also use interdisciplinary theories and approaches such as the auction or game theory to break through the limitation of existing algorithms and get better offloading performance.

3.4.2 MAMoC and the gaps

Unified standards for interacting with both Host Mobile Devices and offloading service providers need to be studied. Currently, other approaches are developing in-house closed frameworks for supporting computation offloading in a unique way which does not satisfy inter-operability. There arises the requirement for unifying these works including framework, communication protocol, and decision-making process. Unification helps to adopt a standard way to promote the industrialization of multisite computation offloading. MAMoC tries to define a standard framework design which depend on inter-operable protocols for communication. These will be discussed and demonstrated in Chapter 5. Moreover, the interaction between different components of the system should be hidden from the users. Currently, only little support is available to cross-platform execution. All the analysed approaches in this chapter are mostly tied to one specific hardware. Supporting different mobile architectures remains a key issue that has not been fully supported. MAMoC is implemented on both Android and iOS mobile platforms.

MAMoC uses a modular service discovery module that can discover Nearby Mobile Devices and Cloudlets in an IP-based networking environment such as WLAN. The local service providers only need to be provisioned and discovered before sending offloading requests to them once. Among other benefits, Zero Configuration Networking protocol has superior API support and extensive documentation. It supports a form of infrastructure-less Domain Name System (DNS) called mDNS, which uses IP multicast [30]. As Bonjour is already a widely-deployed industry

standard, it is a natural choice for zero-configuration use of local context offloading services. For longer-distance surrogates in Remote Clouds, the Host Mobile Device can maintain a simple list of addresses that can be dynamically updated. This unified configuration of surrogates allows the feature set of the surrogates to be universally comparable so that profiling the surrogates can be simplified. This leads to quality of service guarantees, which are important in determining the benefit/cost balance of offloading decisions and adapting the other parts of the software to the usable resources. Furthermore, MAMoC develops an automatic annotation generator to identify offloadable tasks without the need for mobile application developers to specify the offlodable tasks manually. The design and reference implementation of these modules are thoroughly discussed in Chapter 5.

The offloading decision making process is the core of the papers surveyed in this chapter. When researchers include more factors or constraints in their offloading algorithms to meet specific performance requirements, the algorithms need to be improved to be more flexible. MAMoC uses a combination of a heuristic based on device and network profiling information and an offloading decision algorithm based on MCDM to make online decisions. The details are provided in Chapter 4 and evaluated in Chapter 6.

3.5 Summary

This chapter collected the related works of the multisite MCO in the literature. Through the survey methodology and using our inclusion/exclusion criteria, 20 primary studies were selected for further investigation to derive the taxonomy. An in-depth study of the works in each taxonomy classification were shown. The chapter ended with identifying current trends and research gaps in the literature and how the detailed works of MAMoC fit with the aforementioned gaps.

Chapter 4

System Analysis and Models

4.1 Overview

The objective of this work is to propose the design of a generic mobile offloading model that satisfies the hypotheses provided in Chapter 1. This chapter provides an overview of the design requirements, the task offloading models and participating nodes and formulates the problem of offloading to multiple offloading sites in terms of execution time and energy consumption. It further explains the offloading policy and decision making algorithms that are core parts of the offloading decision engine. Additionally, it describes MCDMs that are used to evaluate and rank the candidate offloading sites.

This chapter starts by outlining both functional and non-functional requirements in Section 4.2 to design the aforementioned model and build a prototype of it. Section 4.3 introduces the types of offloading sites that are involved in designing the multisite mobile cloud offloading system and formally defines them to be later used in the design, implementation, and evaluation of the proposed framework. The offloading algorithms are presented in Section 4.4 which decides on the offloading destinations for each particular offloadable task. Section 4.5 describes the multi-criteria decision methods used to evaluate, select, and rank the available offloading sites.

4.2 Requirements Analysis

Before designing the system, a number of functional and non-functional requirements need to be considered. This section uses the taxonomy presented in **??** to furnish a set of requirements which delineate an idealised mobile cloud offloading system. These requirements describe properties which were found to be beneficial

throughout the survey and contemporary literature surveys [44][38][76]. These requirements are used to guide the design and development of MAMoC, our mobile cloud offloading system which is designed in **??** and implemented in **??**.

4.2.1 Functional Requirements

- **Offloading compute-intensive tasks:** mobile applications that contain heavy tasks need to be leveraged by cloud resources. The framework needs to support the execution of mobile applications that have insufficient resources for execution on the mobile device through computation offloading. The computation-intensive tasks take a heavy toll on the device's battery power and computing resources and should therefore be offloaded to proximate and more powerful offloading sites.

- **Supporting local and remote executions:** the mobile application should remain fully functional in the absence of cloud computing support. This can be achieved through partial task offloading and the awareness of the offloading decision engine.

- **Discovering offloading sites:** due to the dynamic nature and potential mobility of offloading sites in dynamic environments, mobile devices need to be able to discover Nearby Mobile Device, Cloudlets, and Remote Clouds cloud servers.

- **Selecting the most optimal offloading site(s):** if more than a resource provider is available, the mobile device should offload computation to the site that is likely to return a response in the shortest amount of time, and result in consuming the least energy before the mobile device loses connectivity to it.

- **Managing the offloading sites:** assuming that mobile code offloading sites are always available is not realistic. Therefore, a user-friendly management service should be provided to manage, configure, and deploy surrogates at different levels of the cloud spectrum.

4.2.2 Non-Functional Requirements

- **Performance enhancement:** the offloaded tasks of the mobile application should improve the overall performance and user experience compared to when the application is executed locally.

- **Energy efficiency:** the total energy consumption on the mobile device when offloading compute-intensive operations (request, execution, and response) should be less than the energy consumed by local execution.
- **Framework reusability and extensibility:** the development of the framework needs to follow a modular approach to achieve code reusability by other mobile cloud application developers. Adding or removing new functionalities should also be made easy for other framework developers.
- **Simplicity and ease of deployment:** the client framework should be made available as a library plugin to be used by application developers. Incorporating MAMoC into existing Android projects should be quick and easy. The server runtime environment should be automatically deployed to new offloading sites through simple commands.
- **Fault tolerance and reliability:** mobile applications often face changes in runtime environments so that the adaptation on offloading is needed. It is decided at application runtime whether tasks need to be offloaded and which parts should be executed remotely. For instance, if the remote site becomes unavailable due to unstable network connection, the computation executed on it should be brought back to the device or be sent to another available site on the fly.
- **Security and privacy:** the mobile device needs to be secured from the malicious cloud and nearby nodes. The data privacy issues of computation offloading also need to be considered by allowing the users to select trusted offloadees for private data.

4.3 Task Models and Problem Formulation

The system considers a heterogeneous mobile cloud environment, comprising the Host Mobile Device, the other Nearby Mobile Device, the fixed edge servers (Cloudlets) and Remote Cloud servers. In the user level, a mobile device that runs applications seeking opportunities to offload tasks to the mobile cloud infrastructure. Mobile applications have different QoS requirements [152]. For instance, the speed of execution is more important for an application that contains computation-intensive tasks. While energy saving can be of more importance to users who are running out of the battery in their devices.

4.3.1 Compute Nodes

The general design of the offloading system can be supported by a P2P model in local mobile device offloading to Nearby Mobile Devices as well as a client-server model in the offloading to Cloudlet and Remote Cloud servers [44]. The Nearby Mobile Devices are managed as in an unstructured P2P network and all the Host Mobile Devices send the offloading requests greedily [143].

4.3.1.1 Self Node

Self Node is the Host Mobile Device which initiates the task offloading requests in the mobile applications which contain heavy tasks. These can be resource-constrained mobile devices such as wearable devices or lower-end smartphones. It is modelled as a 4-tuple:

$$SN = (CP_{sn}, T, BL_{sn}, BS_{sn})$$

where CP_{sn} is the CPU speed (in MIPS) of the host mobile device, T is the list of the offloadable tasks where T_i is the ith task for $i = 0, 1, 2, ..., n$, BL_{sn} is the battery level (1-100) and BS_{sn} is the battery state (charging = 1 or not charging = 0) of the self node.

4.3.1.2 Mobile Node

Mobile Node is a cellphone, tablet, phablet or any other portable Nearby Mobile Device that can connect to the Internet and is able to process offloading requests and send back results [95]. It is modelled as a 4-tuple:

$$MN = (CP_{mn}, RTT_{mn}, BL_{mn}, BS_{mn})$$

where CP_{mn} is the CPU speed (in MIPS) of a mobile device, RTT_{mn} is the round trip time in milliseconds from Self Node to Mobile Node (Network overhead), and BL_{mn}, BS_{mn} are the battery level and state of the mobile node, respectively.

In some studies, these nodes are considered edge nodes for their closeness to the Host Mobile Device [40]. We separate the edge nodes from nearby mobile devices based on two reasons. First, we consider edge nodes to be more powerful in both computation power and memory. Moreover, they do not suffer from having a limited battery life as lower-end mobile devices. Second, we do not restrict edge nodes to running on the same operating system as the host mobile device. The server runtime environment can be deployed to the edge nodes but that would not be the case for the Nearby Mobile Device which are assumed to be running on the Android operating system similar to the Host Mobile Device.

Table 4.1 Compute Node Notations

Symbol	Description
SN	Self Node (host mobile device)
MN	Mobile Node (nearby mobile device)
EN	Edge Node (edge server or cloudlet)
PN	Public Node (remote cloud server)
CP_{sn}	Computation speed of Self Node
CP_{mn}	Computation speed of Mobile Node
CP_{en}	Computation speed of Edge Node
CP_{pn}	Computation speed of Public Node
RTT_{mn}	Network overhead to Mobile Node
RTT_{en}	Network overhead to Edge Node
RTT_{pn}	Network overhead to Public Node
BL_{sn}	Battery level of Self Node
BS_{sn}	Battery state of Self Node
BL_{mn}	Battery level of Mobile Node
BS_{mn}	Battery state of Mobile Node

4.3.1.3 Edge Node

These are the immobile proximate surrogates usually within a single network hop proximity to Self Node. Proximate connected nodes [80] are also connected at runtime to the Remote Clouds. A synchronous offload operation is performed due to the availability of a high bandwidth connection to the surrogate. The communication between this node and the Remote Cloud is a multi-hop over a high bandwidth connection but communication between the two is unnecessary for every offload operation (i.e., only needed for downloading application package files, data synchronization, or to fetch missing resources for the resource-dependent tasks). It is modelled as a 2-tuple:

$$EN = (CP_{en}, RTT_{en})$$

where CP_{en} is the CPU speed (in MIPS) of the edge node and RTT_{en} is the round trip time in milliseconds from the self node to the edge node.

4.3.1.4 Public Node

Public Node is a Remote Cloud server that represents a computing element in the Internet that can execute services for mobile devices. Similar to an edge node, it is modelled as a 2-tuple:

$$PN = (CP_{pn}, RTT_{pn})$$

where CP_{pn} is the CPU speed (in MIPS) of the public node, and RTT_{pn} is the round trip time in milliseconds from the self node to the public node.

4.3.2 Problem Description

According to the different offloading granularities discussed in **??**, parts of mobile application can be offloaded to external surrogates. We model the offloadable tasks in a mobile application based on their independence (whether they can be partitioned into subtasks) and their resource dependence (whether they need an input file data to be present in the offloading site).

A mobile application can contain multiple tasks T_i where $i = 1, 2, ..., n$. Each task can be expressed as

$$T_i = (ID_{T_i}, D_{T_i}, W_{T_i}, PR_{T_i}, DT_{T_i})$$

where ID is the task identifier, D_{T_i} is the data size of the task i in bits, W_{T_i} is the number of CPU instructions that are required to execute the task, PR_{T_i} is 1 if the task is parallelizable or 0 otherwise, and DT_{T_i} indicates the deadline threshold time for task execution.

It turns into a multi-objective optimization problem when simultaneous minimization of overall makespan for completing the tasks and energy consumption of the resources are taken into account [152].

When a task is offloaded, the mobile user has two main expectations, including reducing the execution time of the task or saving the energy of the mobile device. Next, we will analyse the two parts separately before considering them in the task offloading cost.

4.3.3 Execution Time Analysis

The computational time of executing a task locally on a mobile device depends on the number and type of instructions and processing speed (instructions per second) of the mobile device. The processing power and computational task are generalized in terms of Million Instructions Per Second (MIPS). For the mobile device SN, the computational speed (in MIPS) is CP_{sn} and W_{T_i} is the number of instructions of a computational task (in MI). The processing time of the task locally on the mobile device is denoted in Eq. (4.1)

$$M_{T_i}^{SN} = \frac{W_{T_i}}{CP_{sn}} \tag{4.1}$$

Table 4.2 Computed Variable Notations

Variable	**Description**
T_i	ith task in Self Node
D_{T_i}	Data size of task i
W_{T_i}	Number of instructions needed to execute task i
DT_{T_i}	Deadline Threshold of executing ith task
ET_{T_i}	Deadline Threshold of energy consumed for ith task
$\mathrm{Off}_{\mathrm{T_i}}$	Offloading indicator variable
$M_{T_i}^{SN}$	Makespan of processing ith task locally
$M_{T_i}^{S_j}$	Makespan of processing ith task remotely on site j
$M(X)$	Total makespan of all the offloadable tasks of a mobile application
$E_{T_i}^{SN}$	Execution time of task i when running locally
$E_{T_i}^{S_j}$	Energy consumption of self node when running task i on site j
$E(X)$	Total energy consumption of offloadable tasks of a mobile application
PG_{T_i}	Performance gain of task i
EG_{T_i}	Energy efficiency gain of task i
$Cost(T_i)$	Weighted task offloading cost
$Cost(X)$	Total application offloading cost

The execution time of an offloaded task depends on the size and the number of instructions of the computational task (i.e., code size and its computational intensity), size of any required data, and available network throughput, and the computational speed of the offloading site [76]. The mobile device SN offloads D_{T_i} bits of data to the site S_j. Let CP_{S_j} in CPU cycles per second denote the computation capability of the site S_j.

Because of the proposed application model, it is necessary to consider the amount of time spent on the preparation phase of the computational request for offloading on the mobile device (when the task is offloaded for the first time) and the code transformation phase on the offloading site. They are denoted as W_{pre} and W_{ct} to represent the number of instructions required for each of them, respectively. Thus, the first part of Eq. (4.2) is the data transmission time and the second part is the offloading request preparation time on Self Node. The last two parts are the code transformation time and the task processing time on the offloading site j.

$$M_{T_i}^{S_j} = \frac{D_{T_i}}{RTT_{S_j}} + \frac{W_{pre}}{CP_{sn}} + \frac{W_{ct}}{CP_{S_j} + \frac{W_{T_i}}{CP_{S_j}}} \tag{4.2}$$

The time spent for sending back the result from the sites to the mobile devices is not considered. This amount of time is often very short compared with the request data transfer time and task execution time [96].

For task i, let $\mathrm{Off_{T_i}} \in \{0, 1\}$ be an offloading indicator variable. Let $\mathrm{Off_{T_i}} = 1$ if task i is executed locally or 0 otherwise. The total execution time for all the offloadable tasks of a mobile application can be obtained by:

$$M(X) = \sum_{T_i \in N} (\mathrm{Off_{T_i}} \times M_{T_i}^{SN} + (1 - \mathrm{Off_{T_i}}) M_{T_i}^{S_j}) \tag{4.3}$$

4.3.4 Energy Consumption Analysis

Ideally, a task should be offloaded only when the amount of energy consumed to execute the task on the mobile device is greater than the amount of energy consumed to execute it remotely. In other words, if executing a program locally consumes more energy than remote execution, then the offloading technique is a better solution. If a task is executed on the mobile device, only the device's energy is spent during execution. Therefore, the energy required to locally execute a task includes the energy consumed by the mobile device during execution. However, if the same task is executed on the offloading site, its code and all the data needed during execution should first be sent to the site; this incurs the consumption of energy during data transmission. Moreover, for task execution on the cloud, a small amount of energy is consumed even though the mobile device is in an idle waiting state. Therefore, the energy required to execute a task on the cloud includes the energy consumed to transmit the code and its related data to the cloud along with the energy consumed by the mobile device while in a waiting state. The amount of energy consumed by the offloading site is not considered in this analysis.

When the task is executed locally, the energy consumption of the device depends on the computational time of the task and processor frequency of the device. The local energy consumption denoted as $E_{T_i}^{SN}$ results from multiplying the respective energy coefficient value of the processor speed $P_{CP_{sn}}$ by the task execution time which was previously modelled in Eq. (4.1).

$$E_{T_i}^{SN} = P_{CP_{sn}} \times \frac{W_{T_i}}{CP_{sn}} \tag{4.4}$$

For the offloaded tasks, the mobile device energy is consumed in terms of computational request preparation, the transmission of the data, radio idle mode (while waiting for the offloaded task to be executed remotely), reception of result and its integration into the application. When task i is offloaded, the energy consumption denoted as $E_{T_i}^{S_j}$ which consists of three parts: the energy required for the task marshalling time of the first instance of offloading denoted as $E_{T_i}^{pre}$,

the energy consumption of waiting for remote execution $E_{T_i}^{wait}$, and the energy consumption of transferring the required data to offloading sites $E_{D_{T_i}}^{trans}$, described as

$$E_{T_i}^{S_j} = E_{T_i}^{pre} + E_{T_i}^{wait} + E_{D_{T_i}}^{trans} \quad = P_{CP_{sn}} \times \frac{W_{pre}}{CP_{sn}} + P_{idle} \times \frac{W_{T_i}}{CP_{S_j}} + P_{tr} \times \frac{D_{T_i}}{RTT_{S_j}} \tag{4.5}$$

Where P_{idle} indicates the waiting power of mobile devices when task i is migrated to site j. P_{tr} denotes transfer power of mobile devices which is the energy consumption rate of the wireless medium used to transfer the data.

It is clear that the transmission energy used to upload each task will depend on the network transmission bandwidth. Therefore, changes in the wireless network bandwidth will affect the offloading decision. For example, if we assume that transmission time of each task is equal to the size of each task divided by the network transmission rate, then any variation in the transmission rate will affect the final decision of whether to offload this task.

Similar to calculating the total execution time of an application, the energy consumption for all the offloadable tasks of the mobile device is defined as:

$$E(X) = \sum_{T_i \in N} (\mathrm{Off_{T_i}} \times E_{T_i}^{SN} + (1 - \mathrm{Off_{T_i}}) E_{T_i}^{S_j}) \tag{4.6}$$

4.3.5 Task Offloading Cost

The offloading cost can be used as a metric to measure whether to offload the task or execute it locally. The performance gain PG_{T_i} of a task is the difference between executing it on the mobile device and offloading it.

$$PG_{T_i} = M_{T_i}^{SN} - M_{T_i}^{S_j} \tag{4.7}$$

To reduce task execution time, it is only beneficial to offload the task if $PG_{T_i} > 0$.

Similarly, the energy efficiency gain EG_{T_i} of a task is the difference of the energy consumption of the mobile device when the task is executed locally or remotely.

$$EG_{T_i} = E_{T_i}^{SN} - E_{T_i}^{S_j} \tag{4.8}$$

To reduce energy consumption, it is only beneficial to offload the task if $EG_{T_i} > 0$.

For a weighted task offloading cost $Cost(T_i)$, $\lambda_m^{T_i}, \lambda_e^{T_i} \in [0, 1]$ are scalar weights, and $\lambda_m^{T_i} + \lambda_e^{T_i} = 1$. These weights can be adjusted by the preference related with energy and delay deadline of mobile application tasks.

$$Cost(T_i) = \lambda_m^{T_i} \times PG_{T_i} + \lambda_e^{T_i} \times EG_{T_i} \tag{4.9}$$

Overall, the optimization problem formulated is how to select the offloading site to offload for minimizing both the execution time and energy consumption of the task. The optimization framework can be formulated as follows:

$$\min_{C,S} Cost(T_i) \tag{4.10}$$

$$C = M_{T_i}^{S_j} \leq DT_{T_i} \tag{4.11a}$$
$$S = E_{T_i}^{S_j} \leq ET_{T_i} \tag{4.11b}$$

The constraint Eq. (4.11a) denotes the completion time constraint which ensures that the total completion time of the task is bounded by the required maximum finish time (i.e., delay deadline). Similarly, Eq. (4.11b) specifies that energy consumption is less than or equal to the maximum energy consumption.

Finally, the overall application offloading cost $Cost(X)$ can be defined by the summation of the total weighted makespan $M(X)$ and the total weighted energy consumption $E(X)$:

$$Cost(X) = \lambda_m \times M(X) + \lambda_e \times E(X) \tag{4.12}$$

4.4 Offloading Policy

Previous research efforts have been using the MCO approach such as CloneCloud [31], MAUI [35], and ThinkAir [73] due to the capability of an application to either run locally on a mobile device or to offload some of its computationally expensive tasks to be executed outside the mobile device when a connection is available. Since a mobile application can contain parts of the code which must be executed locally (unoffloadable tasks) such as codes that access local sensors (e.g. GPS, camera), it is not recommended to offload the entire application code [50].

Moreover, the task offloading cost (e.g. battery consumption, delay time) may outweigh the offloading benefits. Thus, deciding whether to offload and where to offload are continuous challenges in the task offloading policy.

In constructing a multisite task offloading policy, the single-site scenario offloading where only a single candidate offloading site is available at the time of offloading needs to be considered as well. Deciding whether the task is worth offloading and where to offload the task can be computed under these different circumstances.

For deciding whether the task has to be offloaded, one way is to compare profiling results of the local and remote executions for identifying the offloading cost. However, this creates an extra overhead before every offloading operation [76]. Instead, benchmarking the sites and profiling the network status can be used to calculate an offloading score as described in Subsection 4.4.2.

In an ideal multisite offloading scenario, the Host Mobile Device is connected to more than a site at the time of running the application containing the offloadable tasks. By checking the number of S_j connected candidate offloading sites, the list of sites will be passed to the multi-criteria solver component explained in Section 4.5.

4.4.1 Decision Making Algorithm

The context-aware decision algorithm needs to consider a set of contextual parameters including the network throughput and wireless properties, hardware features of both the Host Mobile Device and connected nodes in the mobile cloud infrastructure. The algorithm needs to decide whether it is beneficial to offload.

With a single offloading site, the node with the highest calculated offloading score is passed to the deployment controller for remote execution. Otherwise, a fuzzy multi-criteria mechanism to decide which resources to use as the offloading locations are used. Finally, the algorithm returns a list of decision pairs containing *[Execution Location: Offloading Percentage]*

The Algorithm 4.1 demonstrates the steps involved in offloading decision making. The illustrated symbols in Table 4.3 are used in the algorithm. The inputs include Self Node modelled earlier, the respective offloadable task T_i, and a list of currently available offloading sites S. As described in the next chapter, the offloading execution database contains all the past offloading records, including both locally LR and remotely RR executed tasks. The checkpoint variables $MaxLE$ and $MaxRE$ are used to ensure that the local and remote executions

Symbol	Meaning
LR	Local Results
RR	Remote Results
MaxLE	Maximum Local Executions
MaxRE	Maximum Remote Executions
ExecLoc	Execution Location
OffPer	Offloading Percentage

Table 4.3 Task Offloading Decision Algorithm Symbols

are still beneficial. Finally, the node scores that were collected and calculated earlier are returned to the decision engine.

Every time an offloadable task is initiated, the engine determines if it is beneficial to offload it. Because of the uncertainties inherent in the mobile environment, the offloading decision takes risk into consideration. If a bad decision has been made, it will also adjust its strategy with new information available.

Algorithm 4.1 Task Offloading Decision Algorithm

Input: SN, T_i, S ▷ Self Node, the offloadable task, list of offloading sites
LR, RR ▷ From past offloading execution database records
MaxLE, MaxRE ▷ Execution checkpoint variables
Output: [ExecLoc: OffPer]

1: **if** $Cost(T_i) < 0$ **then** ▷ According to Eq. (4.9)
2: **return** [SN: 100]
3: **end if**
4: **if** PR_{T_i} AND $\lambda_m^{T_i} == 1$ **then** ▷ For parallizable tasks
5: **return** Score-Partitioner(S) ▷ According to Algorithm 4.3
6: **end if**
7: **for** result in LR **do**
8: **if** result.taskID == ID_{T_i} **then**
9: localExecutions.add(result)
10: **else**
11: remoteExecutions.add(result)
12: **end if**
13: **end for**
14: **if** localExecutions % maxLE == 0 **then** ▷ Max local consecutive executions reached, fall to remote execution
15: **return** MC-Solver(S) ▷ According to Algorithm 4.4
16: **end if**
17: **if** remoteExecutions % maxRE== 0 **then** ▷ Max remote consecutive executions reached, fall to local execution
18: **return** [SN: 100]
19: **end if**

4.4.2 Offloading Score

In calculating the offloading score, we need to collect the device context information in the pre-processing phase to model the offloading decision-making process including the CPU power, round trip time, battery state and level of the connected devices. The benchmark score is calculated after workloads are sent to the devices and the execution results are received. The quicker the tasks are completed, the higher the benchmark score. The workloads measure the instruction performance of the device by performing processor-intensive tasks that make heavy use of integer instructions. Initially, we create two types of workloads: compute-bound and memory-bound. Mandelbrot set [45] of an 800x800 pixels sample is used for

the first type. The Fast Fourier Transform (FFT) [68] is used as a memory heavy workload. The score is calculated based on an average of both runtime scores in GFlops as shown in Eq. (4.13).

$$B = (B_{man} + B_{ftt})/2 \tag{4.13}$$

Since the collected values are on different scales, standardising variables are a necessary process. Standardization (or Z-score normalization) refers to the process of subtracting the mean from the value for each variable, resulting in a mean of zero. Then, the difference between each score and the mean is divided by the standard deviation [111].

The computation power and network overhead to Nearby Mobile Devices are initially collected. If Mobile Node is not currently charging, then the offloading score is deducted by the amount of the battery level of the device. The offloading score is then calculated using the following equation:

$$OS_{MN} = (B_{MN} + CP_{MN}) - RTT_{MN} - (1 - BS_{MN}) \times (100 - BL_{MN}) \tag{4.14}$$

Similar to MAC model, edge and public node modelling is a summation of their respective benchmark score, computation speed subtracted by the data transfer cost.

$$OS_{EN} = (B_{EN} + CP_{EN}) - RTT_{EN} \tag{4.15}$$

$$OS_{PN} = (B_{PN} + CP_{PN}) - RTT_{PN} \tag{4.16}$$

MAMoC collects the offloading scores of the local device running the mobile application and all the connected service providers. Algorithm 4.2 shows the process of collecting individual offloading scores calculated and received earlier to generate a dictionary of nodes and their corresponding offloading scores.

Algorithm 4.2 Aggregating offloading scores of the nodes

Input: SN, S
Output: nodeScores

```
1: nodeScores = [:] ▷ A dictionary of nodes and their respective offloading scores
2: localScore = getSelfNodeScore()
3: nodeScores.add(SN, localScore)
4: if S is not empty then
5:     for S_j in S do
6:         score = S_j.getScore() ▷ According to Eq. (4.14), Eq. (4.15), Eq. (4.16)
7:         nodeScores.add(S_j,score)
8:     end for
9: end if
10: return nodeScores
```

The node scores will then be sent to the task partitioning algorithm shown in Algorithm 4.3 to calculate the final task partitioning percentage (offloading percentage) for any given task.

Algorithm 4.3 Task partitioning algorithm using offloading scores

Input: nodeScores ▷ According to Algorithm 4.2
Output: partitioningResult ([ExecLoc: OffPer])

```
1: function SCORE-PARTITIONER(S)
2:     partitioningResult = [:]  ▷ A dictionary of nodes and task allocation percentages
3:     for (node, score) in nodeScores do
4:         partition = (score / totalScore) * 100
5:         partitioningResult.add(node,partition)
6:     end for
7: return partitioningResult
8: end function
```

4.5 Multi-criteria Solver

The solver is a component in the decision engine module that decides on the offloading destinations for the tasks. It receives a list of available sites from the Service Discovery component with all the necessary metadata from the Profilers component. All these components and their interaction will be described in detail in the next chapter. The solver uses the concept of MCDM for evaluating

the offloading criteria and generating a site ordering from the most to the least optimal under different mobile contexts. MCDM is a well-known approach in mobile cloud offloading studies [11].

In this section, the selected offloading criteria are evaluated and validated using one of the MCDM approaches in Subsection 4.5.1. To achieve multiple offloading objectives, group decision making methodology is used which is demonstrated in detail in Subsection 4.5.2. After evaluating the criteria, the pairwise comparison matrix will be passed to the site ranking system to calculate the order of the offloading sites, which is explained in Subsection 4.5.3.

4.5.1 Criteria Evaluation

The choice of selecting a candidate site for computation offloading is more straightforward when considering only one factor, such as decreasing execution time. However, other Quality of Service (QoS)-based criteria such as current bandwidth between Host Mobile Device and the site and the computation power of the sites as well as availability, security, and the monetary costs of the sites need to be considered when making offloading decisions. Five offloading impact factors are presented to show the construction and operation of the offloading model. This model can be easily extended to incorporate new factors to make offloading decision making more comprehensive.

Bandwidth depends on the quality of the wireless connection between the mobile device and the offloading sites. It is shown in previous work in the literature [77] that offloading is not beneficial when the wireless connection is poor due to high wastage of energy of the mobile devices. Speed is an important criterion that compares the speed of the offloading site with that of the mobile device. Failures may occur due to the mobility of mobile devices and unstable connectivity of wireless communication; Thus, availability also affects the offloading outcome. Security and privacy are important concerns for mobile users, especially if the credibility of the offloading site is unknown. Security and privacy are two crucial concepts that need to be maintained during the offloading process. The financial cost is another factor that needs to be considered when comparing different offers from multiple service providers. The operations of computation offloading and data transfer between cloud resources incur additional costs on end-users. Therefore, economic factors should be taken into consideration while making offloading decisions.

Intensity of importance	Definition
1	Equal importance
2	Equal to moderately importance
3	Moderate importance
4	Moderate to strong importance
5	Strong importance
6	Strong to very strong importance
7	Very strong importance
8	Very to extremely strong importance
9	Extreme importance

Fig. 4.1 Standard AHP Comparison Scale

4.5.1.1 AHP Pairwise Comparison

In this work, a popular MCDM approach called Analytic Hierarchy Process (AHP) [116] is used for determining the relative importance of a set of alternatives in a multi-criteria decision problem. It converts the evaluations to numerical values that can be processed and compared and derives a numerical weight or priority for each element of the hierarchy. The criteria will be compared as to how important they are to the decision makers, with respect to the goal. This multi-criteria technique incorporates the intangible aspects associated with the human factor through the use of pairwise comparisons. The results of the pairwise comparison on n criteria can be expressed in an evaluation matrix A illustrated below:

$$\mathbf{A} = (a_{ij})_{n\times n} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, a_{ii} = 1, a_{ji} = 1/a_{ij} \tag{4.17}$$

Offloading decisions involve different qualitative and quantitative parameters [11]. As described earlier, the five chosen decision criteria for selecting offloading sites are bandwidth, speed, availability, security, and price denoted as *C1*, *C2*, *C3*, *C4*, *C5* respectively. The priory of importance depends on what we care about most in the selection of an offloading site. Bandwidth is considered the most significant since it decides the extra communication cost between mobile devices and the sites [12]. Speed also has high importance because it saves execution time which leads to better user experience and less local energy consumption. We assume the priory of importance is ranked as: bandwidth > speed > availability > security > price. However, the priority of these five criteria can be varied in other situations as described in Subsection 4.5.2. The relative performance of each

Table 4.4 Pairwise comparison matrix (A) for offloading criteria

	Bandwidth	**Speed**	**Availability**	**Security**	**Price**
Bandwidth	1	1	5	7	9
Speed	1	1	5	6	8
Availability	1 / 5	1 / 5	1	3	3
Security	1 / 7	1 / 6	1 / 3	1	2
Price	1 / 9	1 / 8	1 / 3	1 / 2	1

criterion against another can be evaluated in a pair-wise manner according to the rubric in Figure 4.1 with 1 being equal and 9 being much better. As a result, the pairwise comparison matrix is generated as shown in Table 4.4.

Using Row Geometric Mean prioritization Method (RGMM) [34], the final weighting results for each criteria $w = (w_1, w_2, ..., w_n)$ are: Bandwidth (C1): 0.4072, Speed (C2): 0.3885, Availability (C3): 0.1083, Security (C4): 0.0572, Price (C5): 0.0384.

4.5.1.2 Consistency Check

The purpose of the consistency check is to test the coordination of the importance degree between each criterion. This is used to avoid the contradiction situation, i.e. for a user, A is more important than B, B is more important than C, and C is more important than A. For example, it is inconsistent in the case where a Decision Maker (DM) thinks availability is strongly more important than security, and security is more important than price, however, the price is more significant than availability in the same context.

A consistency index to measure individual matrix consistency, namely the Geometric Consistency Index (GCI) is developed by [3]. Let $A = a_{ij}$ be a judgement matrix, and let $w = (w_1, w_2, ..., w_n)$ be the priority vector derived from A using the RGMM. The Geometric Consistency Index (GCI) is given below:

$$GCI\,(A) = \frac{2}{(n-1)\,(n-2)} \sum_{i<j} \left(log\,(a_{ij}) - log\,(w_i) + log\,(w_j)\right)^2 \tag{4.18}$$

If $GCI(A^k) = 0$, we have a fully consistent matrix. $\overline{GCI}$ is also provided by [3] for GCI: $\overline{GCI} = 0.31$ for $n = 3$; $\overline{GCI} = 0.35$ for $n = 4$; $\overline{GCI} = 0.37$ for $n > 4$. When $GCI\,(A) < \overline{GCI}$ the matrix A is of acceptable individual consistency. Using Eq. (4.18), the calculated $GCI\,(A)$ for the pairwise matrix is 0.099 which is much less than 0.37 for $n = 5$.

4.5.2 AHP Group Decision Making

In order to enable more fine-grained offloading decision for mobile applications with the different quality of services, the application developers can define the criteria importance for their applications by giving relative weights or select from pre-defined DMs. This work constructs 5 pre-defined DMs to represent different weights for given circumstances of criteria importance using the concept of AHP Group Decision Making (GDM) (AHP-GDM) [115]. The evaluation of the proposed AHP-GDM is conducted using the methodologies suggested in [39] where both the consensus measure and the consensus model are used with a row geometric mean prioritization method.

In modelling the AHP-GDM, let $DM = DM_1, DM_2, ..., DM_n$ be the set of DMs, and $\pi = \pi_1, \pi_2, ..., \pi_m$ be the weight vector of DMs, where $\pi_k > 0, k = 1, 2, ..., m$. The judgement matrix is modelled as $A^{(k)} = a_{ij}^{(k)}$ provided by the decision maker $DM_k (k = 1, 2, ..., m)$. The individual priority vector $w^{(k)} = (w_1^{(k)}, ..., w_5^{(k)})$ is then derived from judgment matrix $A^{(k)}$ using RGMM.

As illustrated in Table 4.5, these pre-defined five DMs are selected according to the runtime requirements of a mobile application. The user can specify weights for each one of them based on the application context. By default, the normal execution which was described in the previous subsection is used if no weights are specified. The following is a brief description of each DM:

1. **Normal Execution (DM1):** this is the default DM which is used for the single decision maker as described in Subsection 4.5.1. Normally, the weight of this DM is set to 1 while the weight of other DMs is set to 0. However, we make some exceptions when the battery level is less than 20% and the battery is currently not charging. We modify the weights according to Table 4.6.

2. **Reduce energy consumption (DM2):** this DM can be given the highest priority when the phone battery is in critical conditions, e.g., less than 20% remaining. Prolonging battery life is one of the most important factors to consider when offloading mobile tasks to external entities.

3. **Reduce execution time (DM3):** due to the resource constraints of mobile devices, we want to increase the response time of the heavy tasks of the mobile applications by offloading them to more powerful surrogates. This DM gives the highest priority to reducing the execution time by giving the highest value to the computational power of the surrogates.

4. **Increase security (DM4):** this DM gives the utmost importance to the security and privacy of the offloading sites. If there are some sensitive and private data which can not be protected enough in a Remote Cloud server, i.e. no authentication or encryption mechanism exist between the Host Mobile Device and the connected server, then the likelihood of selecting that particular site decreases. It is necessary to use certain encryption algorithms to the data stored in the offloading sites in order to preserve privacy. Moreover, it is advisable that the encryption algorithms be lightweight in terms of computation.

5. **Reduce the financial cost (DM5):** Using cloud infrastructure resources imposes financial charges on the end-users, who are required to pay according to the Service Level Agreement (SLA) agreed on with the cloud vendor serving them. The overall monetary cost of cloud resources can get expensive as we get numerous requests from mobile users. Finding cheaper offloading sites can benefit application providers for reducing the overall cost of running mobile tasks remotely.

4.5.2.1 Group Consistency Check

For GDM using preference relations (pairwise comparisons), the consistency measure includes two concerns including the individual consistency of a single DM and of a whole group of DMs [39]. At the beginning of each GDM problem, decision-makers' opinions may differ from each other substantially. Consensus models are decision aid tools to help DMs reach the consensus based on the established consistency measure.

In the AHP GDM context, the two techniques traditionally used are (i) Aggregation of Individual Judgements (AIJ) and (ii) Aggregation of Individual Priorities (AIP) [47]. On one hand, AIJ uses the weighted geometric mean method to aggregate individual judgement matrices to obtain a collective matrix. Then, prioritization methods, such as the row geometric mean method, are used to derive a priority vector to order a collective judgement matrix. On the other hand, AIP uses the weighted geometric mean method to aggregate individual priorities derived using prioritization methods to obtain the best alternative(s).

The group judgement matrix and group priority vectors can be produced in the following: the priority vectors are first obtained for each individual, $w_{(k)} = w_i^{(k)}$ and k = 1,...,r, using Weighted Geometric Mean Method (WGMM) and then aggregated to obtain the priorities of the group using:

	B	**Sp**	**A**	**Sc**	**P**
B	1	1	5	7	9
Sp	1	1	5	6	8
A	1 / 5	1 / 5	1	3	3
Sc	1 / 7	1 / 6	1 / 3	1	2
P	1 / 9	1 / 8	1 / 3	1 / 2	1

(a) A^1: Energy efficient and fast response (normal execution)

	B	**Sp**	**A**	**Sc**	**P**
B	1	5	7	9	9
Sp	1 / 5	1	3	7	7
A	1 / 7	1 / 3	1	5	5
Sc	1 / 9	1 / 7	1 / 5	1	1
P	1 / 9	1 / 7	1 / 5	1	1

(b) A^2: Reduce energy consumption

	B	**Sp**	**A**	**Sc**	**P**
B	1	1 / 7	1 / 5	7	7
Sp	7	1	3	9	9
A	5	1 / 3	1	9	9
Sc	1 / 7	1 / 9	1 / 9	1	1
P	1 / 7	1 / 9	1 / 9	1	1

(c) A^3: Reduce execution time

	B	**Sp**	**A**	**Sc**	**P**
B	1	1	1	1 / 9	3
Sp	1	1	3	1 / 9	3
A	1	1 / 3	1	1 / 9	3
Sc	9	9	9	1	9
P	1 / 3	1 / 3	1 / 3	1 / 9	1

(d) A^4: Increase security

	B	**Sp**	**A**	**Sc**	**P**
B	1	1	3	3	1 / 9
Sp	1	1	3	3	1 / 9
A	1 / 3	1 / 3	1	3	1 / 9
Sc	1 / 3	1 / 3	1 / 3	1	1 / 9
P	9	9	9	9	1

(e) A^5: Reduce the financial cost

Table 4.5 The judgement matrices from the decision makers based on the application requirements. B: Bandwidth, Sp: Speed, A: Availability, Sc: Security, P: Price

Table 4.6 Decision maker weights under different battery context

Battery	DM Weight
BL > 80 or Charging	(0,0,1,0,0)
80 > BL > 20	(1,0,0,0,0)
BL < 20	(0,1,0,0,0)

Table 4.7 The GCI and $GCCI$ values of $A^k(k = 1, 2, ..., 5)$ and A^G

	$A^{(1)}$	$A^{(2)}$	$A^{(3)}$	$A^{(4)}$	$A^{(5)}$	$A^{(G)}$
$GCCI$	0.099	0.3064	0.3451	0.3192	0.243	0.2837

$$w_i^{(G)} = \prod_{k=1}^{r} \left(w_i^{(k)}\right)^{\pi_k},\ i = 1, \ldots, n \tag{4.19}$$

The group judgement matrix $A^G = a_{ij}^{(G)}$ is then calculated by:

$$a_{ij}^{(G)} = \prod_{k=1}^{r} \left(a_{ij}^{(k)}\right)^{\pi_k},\ i, j = 1, \ldots, n \tag{4.20}$$

The Geometric Cardinal Consensus Index (GCCI) of A^k is determined through the following:

$$GCCI\left(A^{(k)}\right) = \frac{2}{(n-1)(n-2)} \sum_{i<j} \left(log\left(a_{ij}^{(k)}\right) - log\left(w_i^{(k)}\right) + log\left(w_j^{(k)}\right)\right)^2 \tag{4.21}$$

If $GCCI(A^k) = 0$, then the kth decision maker is of fully cardinal consensus with the collective preference. Otherwise, the smaller the value of $GCCI(A^k)$, the more the cardinal consensus. According to the actual situation, the decision makers establish the thresholds $\overline{GCCI}$ for $GCCI(A^G)$.

If $GCCI\left(A^{(G)}\right) \leq \overline{GCCI}$, we conclude that the acceptably cardinal consensus are reached among the decision makers.

According to [39], $\overline{GCCI}$ for n = 5 is 0.35, as listed in Table 4.7 $GCCI\left(A^{(k)}\right) < \overline{GCCI}$ for (k = 1,2,...,5), so $A(^1)$,...,$A(^5)$ are of acceptably individual consistency so as $GCCI(A^G)$ having an acceptable group consistency.

4.5.3 Site Ranking Calculation

Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) is one of the most widely used techniques of MCDM [78]. This technique has been chosen for its lightweight processing and faster execution time compared to other MCDMs [146]. It is able to maintain the same number of steps regardless of the number of attributes which makes it ideal for using it in resource-constrained mobile devices. Alternatives are evaluated according to a set of criteria received from AHP matrix in a TOPSIS technique.

This technique is based on the principle that the selected alternative, i.e. offloading site S_i, should have the least distance to the positive ideal and the most distance to the negative ideal. For each criterion considered previously in

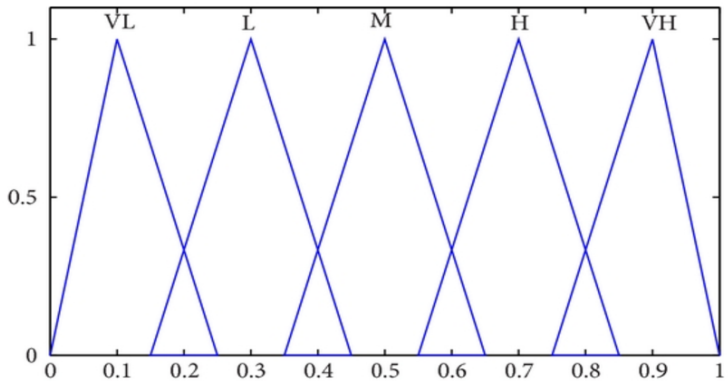


Fig. 4.2 Fuzzy linguistic terms mapped to their numerical range values

AHP, a set of linguistic variables is used to define the importance weights of them. These criteria weights need to be assigned for each alternative, i.e. offloading sites (Nearby Mobile Devices, Cloudlets, and Remote Clouds) per the availability.

Initially, a subjective evaluation is set on a rating of alternatives and the importance weight of criteria to produce the fuzzy decision matrix D. The numerical range values are shown in Figure 4.2 for each fuzzy linguistic variable [21].

$$\mathrm{D} = \begin{bmatrix} \tilde{x}_{11} & \tilde{x}_{12} & \dots & \tilde{x}_{1j} & \tilde{x}_{1m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{x}_{n1} & \tilde{x}_{n2} & \dots & \tilde{x}_{nj} & \tilde{x}_{nm} \end{bmatrix} \tag{4.22}$$

Where $\tilde{x}_{ij}$, j=1,2,..,m are linguistic variables. These variables are described by triangular fuzzy numbers, $\tilde{x}_{ij} = (a_{ij}, b_{ij}, c_{ij})$ as demonstrated in Table 4.8.

Table 4.8 Fuzzy membership and Linguistic scale for TOPSIS

Linguistic Values	**Fuzzy Ranges**
Very low (VL)	(0.0, 0.0, 0.25)
Low (L)	(0.0, 0.25, 0.50)
Good (G)	(0.25, 0.50, 0.75)
High (H)	(0.50, 0.75, 1.0)
Very High (VH)	(0.75, 1.0, 1.0)

Then, the normalized fuzzy decision matrix of the alternative ratings is constructed in (D) using the linear scale transformation to transform the various

criteria scales into comparable scales [63]. The normalized fuzzy decision matrix for both benefit and cost criteria is obtained by:

$$\tilde{r}_{ij} \simeq (\frac{a_{ij}}{c_j^+},\ \frac{b_{ij}}{c_j^+},\ \frac{c_{ij}}{c_j^+}),\ c_j^+ = \max_i c_{ij}(\text{Benefit criteria}) \tag{4.23a}$$

$$\tilde{r}_{ij} = (\frac{a_j^-}{a_{ij}},\ \frac{a_j^-}{b_{ij}},\ \frac{a_j^-}{c_{ij}}),\ a_j^- = \min_i a_{ij}(\text{Cost criteria}) \tag{4.23b}$$

The weighted normalized decision matrix, $\tilde{v}_{ij}$ is calculated by multiplying the weight of the criteria, $\tilde{w}_j$, by the elements $\tilde{w}_j$ of the normalized fuzzy decision matrix:

$$\tilde{v}_{ij} = \tilde{r}_{ij} \times \tilde{w}_j \tag{4.24}$$

Then, the Fuzzy Positive Ideal Solution $(FPIS, S^+)$ and Fuzzy Negative Ideal Solution $(FNIS, S^-)$ are determined using:

$$S^+ = \{\tilde{v}_1^+, \tilde{v}_j^+, ..., \tilde{v}_m^+\} \tag{4.25a}$$

$$S^- = \{\tilde{v}_1^-, \tilde{v}_j^-, ..., \tilde{v}_m^-\} \tag{4.25b}$$

Where $\tilde{v}_j^+ = (0.75,1.0,1.0)$ and $\tilde{v}_j^- = (0.0,0.0,0.25)$.

Finally, the distances d_i^+ and d_i^- of each alternative $S_i(i = 1, 2, ..., m)$ from FPIS and FNIS are calculated, respectively:

$$d_i^+ = \sum_{j=1}^{n} d_v(\tilde{v}_{ij},\quad \tilde{v}_j^+) \tag{4.26a}$$

$$d_i^- = \sum_{j=1}^{n} d_v(\tilde{v}_{ij},\quad \tilde{v}_j^-) \tag{4.26b}$$

where $d_v(\cdot,\cdot)$ represent the distance between two fuzzy numbers according to the vertex method [21]. Let $\tilde{m} =, (m_1,\ m_2,\ m_3)$ and $\tilde{n} = (n_1, n_2, n_3)$ be two triangular fuzzy numbers. Distance calculation of triangular fuzzy numbers is calculated by the following:

$$d_v(\tilde{m}, \tilde{n}) = \sqrt{\frac{1}{3}[(m_1 - n_1)^2 + (m_2 - n_2)^2 + (m_3 - n_3)^2]} \tag{4.27}$$

Finally, the Closeness Coefficient CC_i is calculated from both ideal d_i^+ and anti-ideal d_i^- distances:

$$CC_i = \frac{d_i^-}{d_i^- + d_i^+},\quad 0 < CC_i < 1,\quad i = 1, 2, ..., n \tag{4.28}$$

The set of alternatives can now be preference ranked according to the descending order of CC_i. A set of numerical examples will be given in Chapter 6 to demonstrate the combination of these MCDM approaches in achieving an optimal ranking of the offloading sites.

The Algorithm 4.4 demonstrates the steps involved in using AHP and fuzzy TOPSIS for multisite offloading in both single and group decision making processes. The algorithm takes as the input the number of DMs and their respective pairwise comparison values. Then, the criteria evaluation is conducted based on the given values to produce the matrix A. In the case of GDM, the judgment matrix needs to be generated from all the DM matrices. Once the AHP produced the A, lines 9-12 of the algorithm generates the site ranking orders and their respective closeness coefficient values which are used as offloading percentages of the offloading site.

The source code of the MCDM program is decoupled from the main decision engine of MAMoC framework for demonstration clarity and reusability of the program for different purposes [1].

Algorithm 4.4 Multi-Criteria Solver Algorithm

Input: DM (Decision Makers)
A = [n][n] ▷ AHP pairwise comparison matrix Eq. (4.17)
PCV_{DM_i} ▷ Pairwise Comparison Values of the DMs
D = [Fuzzy] ▷ Fuzzy linguistic values in Table 4.8
Output: [ExecLoc: OffPer]
1: **function** MC-SOLVER(S)
2: **if** count(DM) == 1 **then**
3: A = calculateAHP(PCV[0]) ▷ According to Table 4.4
4: **else** ▷ multi-objective optimisation using GDM
5: **for** dm_i in DM **do**
6: $GA[i]$ = calculateAHP(PCV[i]) ▷ According to Table 4.5
7: **end for**
A = calculateJudgmentMatrix(GA) ▷ According to Eq. (4.20)
8: **end if**
9: D = calculateFuzzyTopsis(S, A) ▷ Eq. (4.22) - Eq. (4.24)
10: d^+ = calculateIdealDistance(D) ▷ Eq. (4.26a)
11: d^- = calculateAntiIdealDistance(D) ▷ Eq. (4.26b)
12: **return** [ExecLoc: OffPer] = calculateClosenessCoefficient(d^+, d^-) ▷ Eq. (4.28)
13: **end function**

[1] https://github.com/dawand/AHP_Fuzzy_TOPSIS

4.6 Summary

In this chapter, the design requirements, modelling, and decision making algorithms are investigated for offloadable tasks in a mobile application running on a resource-constrained mobile device. The surrounding compute nodes are modelled and considered in both the execution time and energy consumption analysis. The overall task offloading cost needs to be minimised in the optimization problem.

The task offloading policy contains the question of whether a task should be offloaded and where the best offloading destinations are under different mobile contexts. The offloading score is calculated with the collected profiling information and benchmarking mechanisms. The offloading destination decision making is conducted using MCDMs of AHP and fuzzy TOPSIS. The GDM methodology is also used for conducting multi-objective decision making according to different DMs.

Chapter 5

Design and Implementation

5.1 Overview

This thesis proposes Multisite Adaptive Mobile Cloud (MAMoC), which is a MCO framework that offloads Tasks in a mobile application to multiple offloading sites. As shown in Chapter 4, each task offloading decision is made based on the energy and completion time estimations. MAMoC is capable to adaptively recalculate these estimations accordingly after multiple local and remote executions of the task. One of the most important features of the system is that it does not require changes in the operating system in which the mobile application is running [96]. Neither, it needs a specialised mobile operating system in the server-side (Android x-86 [7]) described in Chapter 2. This allows developers to seamlessly adapt the framework to their current Android applications.

The design assumptions of the framework are listed in Section 5.2 that specify the attributes of the tasks that developers need to follow as well as the assumptions for both the communication and execution component designs. Section 5.3 explains the system architecture with a detailed look into both the mobile device and offloading site modules. Section 5.4 discussed the communication services and components used in managing the message flow between Host Mobile Device and Nearby Mobile Devices in device-to-device communications and between Host Mobile Device and ENs and PNs in device to server communications. Section 5.5 describes the task execution workflow starting from the task marshalling on the mobile device up to the point of receiving the executed results back.

The proposed framework is capable of profiling, analysing, and executing (locally or remotely) a task in the running mobile applications. This chapter also discusses the implementation of the components. Each component in MAMoC Client is implemented in a module that provides a standard interface that defines

a set of functionality that can be invoked. This chapter also provides a reference implementation of the server runtime environment (MAMoC Server) deployed to the offloading sites. This reference implementation is divided into two main sections: Section 5.6 describes the concise implementation details of the client framework running on SNs and MNs. While, Section 5.7 explains the server runtime environment components running in ENs and PNs.

5.2 Design Assumptions

The research conducted in this thesis rests on the assumption that the mobile cloud offloading infrastructure belongs to a single mobile user. This is a reasonable assumption in designing a framework which can be used by many mobile applications in the same mobile device. Alternative approaches exist where multiple user offloading requests are considered in the study [20].

The following assumptions are made about the offloadable tasks, the platform execution and how the communication between different nodes in the framework are performed.

5.2.1 Task Specifications

As shown in the works examined in Chapter 3, the partitioning granularity differs from offloading the whole app to the offloading destinations or only a part of the app in the forms of a component [153], a class [131], a method [73], or a thread [31]. In MAMoC, the partitioning is performed in the task level which could represent both a class or a method in a class.

The following assumptions are imposed by MAMoC for the classes that are made offloadable by the application developer:

1. The class is annotated with @Offloadable with or without the optional parameters (parallalizable or resource dependent).
2. The class contains a constructor that receives the required parameters from the caller classes.
3. The class contains a *run* method of either a return or void type.
4. The *run* method does not depend on any native features of the mobile device and does not call any Android system library methods.
5. Similarly, in the case of the method being annotated with @Offloadable, no native feature must exist inside the body of the method.

5.2.2 Communication Assumptions

1. *Device-to-Device communications*

    1.1. Self Node can establish a connection with Nearby Mobile Devices (MNs) using either Wi-Fi P2P or infrastructural Wi-Fi. Most modern Android devices have WiFi-Direct built in them, as described subsequently in **??**. However, the support for incompatible devices is further accomplished by utilising the GO device as an AP [19].

    1.2. The Nearby Mobile Devices (MNs) do not route requests to other devices, i.e MAMoC only supports D2D offloading. The routing problem is not addressed in our implementation, but it would not be difficult to incorporate an existing P2P routing protocol to the framework [93].

    1.3. The Nearby Mobile Devices (MNs) are willing to share a portion of their computational capabilities as a stem of different incentives ranging from their willingness to share resources as in volunteer computing scenarios [91]. An incentive mechanism needs to be considered especially for users with limited battery mobile devices [162].

2. *Device-to-Server communications*

    2.1. Host Mobile Device (Self Node) can connect to Cloudlets (ENs) via infrastructural Wi-Fi by scanning the LAN. The ENs should advertise their local IP address through an Avahi daemon [1] by joining a particular Multicast IP Address as a listener to help with the discovery mechanism.

    2.2. Host Mobile Device (Self Node) can connect to Remote Clouds (PNs) via Wi-Fi or mobile data networks such as 3G/4G. The IP addresses of the PNs are already made available to the mobile device before initiating the framework.

5.2.3 Execution Assumptions

1. MAMoC assumes that the Nearby Mobile Devices (MNs) are provisioned with the same MAMoC-enabled mobile application library (MAMoC Client) before receiving offloading requests.

2. MAMoC assumes that the Cloudlets (ENs) and Remote Clouds (PNs) are provisioned with the MAMoC Server running environment before receiving offloading requests.

[1] http://avahi.org/

3. MAMoC initially assumes that offloaded computation already resides on the offloading site by calling the RPC-based task ID (ID_{T_i}). The task source code and data resources are either stored on the site via static analysis and bytecode refactoring tools at the deployment or obtained the source code from the mobile device at runtime.

5.3 Architecture Overview

In this section, an insight into the architecture of MAMoC is given to address the importance of having a multisite and adaptive task offloading in a heterogeneous mobile cloud environment. The mobile device requesting the offloading service is regarded as a client (Self Node or Host Mobile Device as used throughout this thesis). The proposed system leverages three types of cloud resources: Nearby Mobile Device (Mobile Node), nearby fixed edge nodes or Cloudlets (Edge Node), and Remote Cloud instances (Public Node). These compute nodes were previously described in Subsection 4.3.1. This section also provides a brief description of each module in both MAMoC Client and MAMoC Server programs. The implementation details of the modules are left for the next chapter.

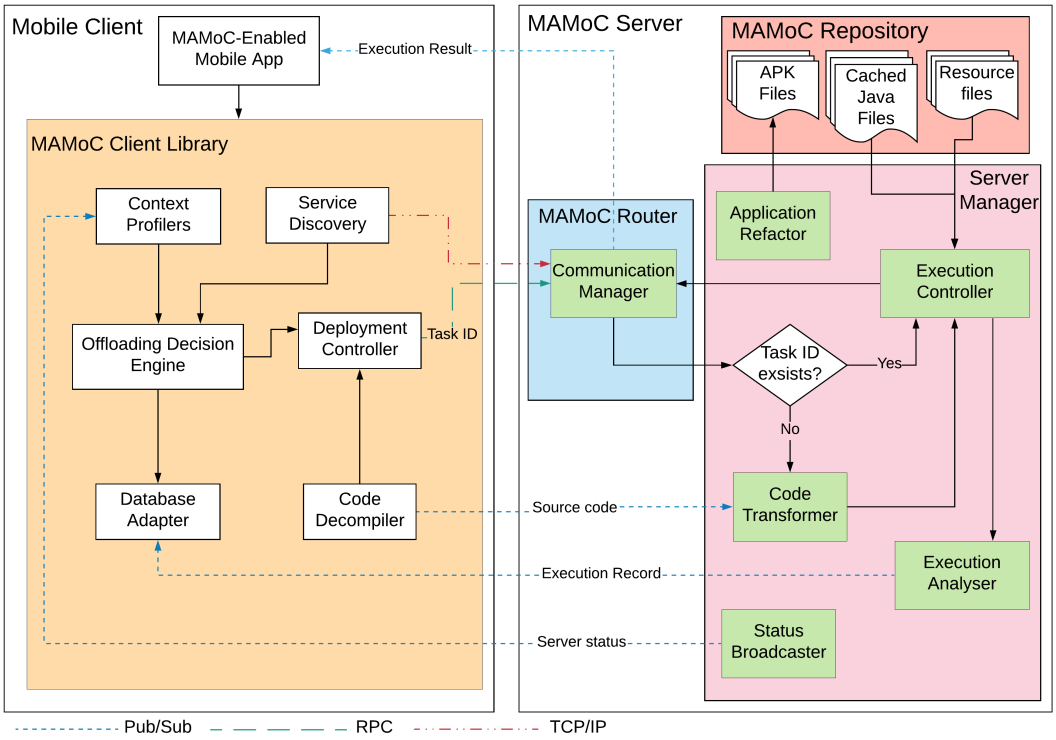


Fig. 5.1 High-level architecture and communication mechanisms of MAMoC

1. **MAMoC Client:** the offloading library that runs on the Host Mobile Device as a built-in library of the MAMoC-enabled mobile applications.

   MAMoC Client contains the following modules:

   1.1. **Context profilers:** profiling is the process of gathering contextual information about devices and programs to help offloading making decisions. This information can be the computation and communication costs of application units (program profiler), the device-specific characteristics such as battery state and level, computation and memory capabilities (device profiler) and wireless and medium channels characteristics (network profiler). Since profiling incurs additional overhead and energy consumption, we adopt an on-demand monitoring strategy. We only fetch context data when the offloadable methods are invoked. All the collected data will be fed to the decision engine to be used in the offloading decision making algorithms.

   1.2. **Service discovery:** for the mobile device to leverage nearby and remote resources, it needs to know where they are located by saving their network address (IP addresses or URLs). MAMoC maintains a list of all the potential resources with their network addresses once they are discovered and a successful connection is established. The Host Mobile Device sends periodic pings to the connected nodes to ensure their availability to serve the offloading requests.

   After the framework is initialized, service discovery is performed. The hosting mobile device advertises itself in the LAN and searches for available Nearby Mobile Devices. We will explore in detail the mechanisms that we use in Section 5.6.1.1. The Host Mobile Device also scans the local network to detect the running containers on the local Cloudlets (ENs). The Remote Cloud instances are discovered using their IP addresses to open a TCP socket communication channel for offloading requests.

   1.3. **Offloading decision engine:** the offloading decision making is a two-fold process: the engine first checks if it is worth offloading using the decision making algorithm described in Subsection 4.5.1 and if yes, where should the task be offloaded among the multiple available offloading sites as explained in Subsection 4.5.3.

   1.4. **Deployment controller:** after it is decided to offload a task, the task ID and offloading sites information is passed to the deployment

controller to communicate with the router component in MAMoC server. If the task ID does not exist in the server-side, the command is passed to code decompiler which is described next.

It receives the task offloading information such as the class name, method name, decompiled source code, offloading site, resource names, and any extra parameters from the decision engine. It also collects node information from the service discovery if the task is to be offloaded otherwise a local execution will be performed.

1.5. **Code decompiler:** the bytecode of the mobile application binary is decompiled into Java source code to be sent to the offloading site for execution purposes. This is only performed once per task as the source code will be cached in the server side for future executions of the same task by the same or different mobile user requests.

1.6. **Database adapter:** this module records all the local and remote executions on the self node and offloading sites. It contains information about the offloadable task, execution time, energy consumption, offloading site, etc. This information is vital in the offloading decision making algorithm.

2. **MAMoC Server:** MAMoC server consists of three separate interconnected modules running on three containers:

   - **MAMoC Router:** this contains the communication manager component that handles network communications with mobile devices. It also validates the incoming requests from mobile applications.
   - **Server Manager:** most of the server components are deployed on this container. It receives incoming offloading requests from the routers. After executing the task, the execution results will be published.
   - **MAMoC Repository:** this contains the cached resources from previous executions including the source code and resource files. It also contains the fetched Android application executables from Play Store.

   The following are the server-side components:

   2.1. **Communication manager:** this accepts the offloading requests from the self nodes. The requests need to be validated through an authentication scheme, which is described at the end of this chapter.

2.2. **Execution controller:** this module is the core of the MAMoC server, which contains a Java Virtual Machine (JVM) running environment. It receives the task ID from the router. Then, it checks if an existing Java source code already exists in the MAMoC repository. Otherwise, it fetches the source code from the code transformer, which runs some filtering and instrumentation on the decompiled Android code. Finally, it passes the execution result to the communication manager component and execution details to the execution analyser component.

2.3. **Code transformer:** this module uses filtering and instrumentation techniques to transform an Android-based Java code to a Java code that can be run on the JVM bundled in the execution controller component.

2.4. **Application refactor:** given an APK of an Android mobile application, this component analyses it to generate a method call graph and predict the methods which can be marked as offloadable. It also identifies the unoffloadable parts of the application using the Android packages and libraries to find mobile native features. If the mobile application has not been annotated by the developer, the application ID will be provided to the server. The MAMoC repository fetches the APK from the Google Play Store [2] using the app ID. It then runs a static analysis and bytecode refactoring to find the offloadable tasks.

2.5. **Execution analyser:** after the remote execution is complete or unsuccessful, the execution details are given to this component to generate an execution record. This record is then passed to the database adapter in Self Node to aid with future offloading decision making.

2.6. **Status broadcaster:** this component is responsible for monitoring the CPU and available memory of the site. It broadcasts the results to the SNs that have already subscribed to listen to the hardware changes of this particular site.

[2]https://play.google.com/store/apps

5.4 Service Discovery

Service discovery module maintains a list of available services and handles the connections between the Host Mobile Device and mobile cloud infrastructure nodes including the available Nearby Mobile Devices, Cloudlets, all the way up to Remote Cloud servers. The device profiling technique is triggered after a new node is discovered to obtain new profiling information or update existing ones in the *Nodes* table in the helper database. The register of accessible sites is passed to offloading decision engine and later to deployment controller for dealing with the data transfer such as publishing source code of the offloaded tasks, calling the remote procedure calls, and transferring the input files essential for the resource-dependent tasks. There might be a situation where the Nearby Mobile Device is less powerful than the Host Mobile Device so it is not consistently beneficial to offload computation assuming that the remote device is more computationally capable than the local device.

The Nearby Mobile Devices can be leveraged for collaborative sharing of resources to gain utilitaristic benefit. Designing energy-efficient and high bandwidth direct D2D communications is a lasting challenge confronted by mobile platforms [37]. Wi-Fi is still claimed to be the dominant future D2D solution for mobile device connectivity and therefore supports wearable aggregation nodes [66]. One approach is to form a MAC [159] over the LAN based on either zero configuration network technology [30] or in a direct peer communications using Wi-Fi P2P technology [37]. Nevertheless, the discovery via network interfaces can potentially cause additional detection overhead and energy consumption. To avoid this overhead, the module applies a periodic detection strategy, which asynchronously searches for the available devices at certain intervals periodically. The detailed implementation of the service discovery is described in Subsection 5.6.1.

Subsection 5.4.3 discusses the authentication mechanism for security concerns in various mobile network architectures.

5.4.1 Device to Device

The two different communication mechanisms explained in Chapter 2 are employed between the Host Mobile Device and other participated nodes in MAMoC. The zero configuration network and multicast DNS which is used for discovering the locally available resources (MNs and ENs). In the absence of an infrastructural network. The Wi-Fi P2P for enabling ad hoc communications between the Host Mobile Device and nearby mobile equipments are also explained.

1. **Zero Configuration Network**

   This mode of communication can be used for discovering both MNs and ENs. As depicted in Figure 5.2, the Edge Node contains an Avahi daemon which advertises the IP address of the Edge Node in the LAN.

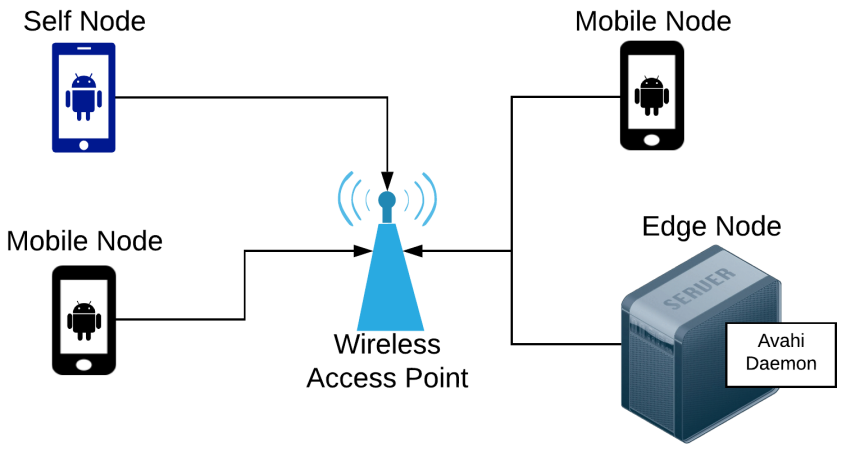


Fig. 5.2 Zero Configuration Network communication diagram

2. **Wi-Fi P2P** Wi-Fi Direct is only available in selected modern Android devices. In MAMoC, using APs in infrastructural Wi-Fi using Zero Configuration Network technology for D2D communications is also utilised. Section 5.6.1.1 provides the implementation details of both approaches with similarities and differences between them. Subsection 5.6.1 describes the implementation details of both approaches on the Android platform. A possible nearby communication setup using Wi-Fi Direct is shown in Figure 5.3.

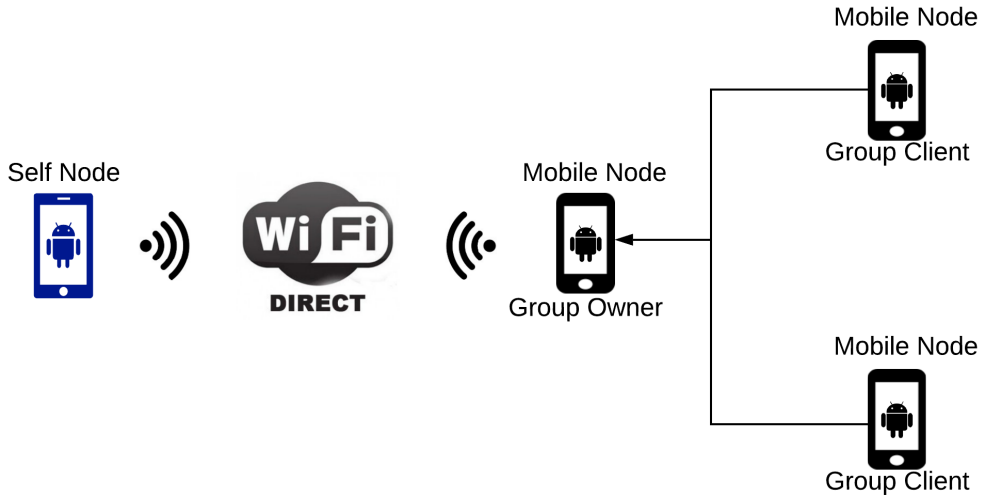


Fig. 5.3 Wi-Fi P2P communication diagram

5.4.2 Device to Server

This section describes how the protocol is applied for managing the communications between Self Node and edge or public nodes in the form of the device to server communications. MAMoC uses WAMP which is a subprotocol of WebSocket for our message communications and we use both RPC and Publish/Subscribe technologies. Next, both message patterns are explained and how MAMoC utilises them.

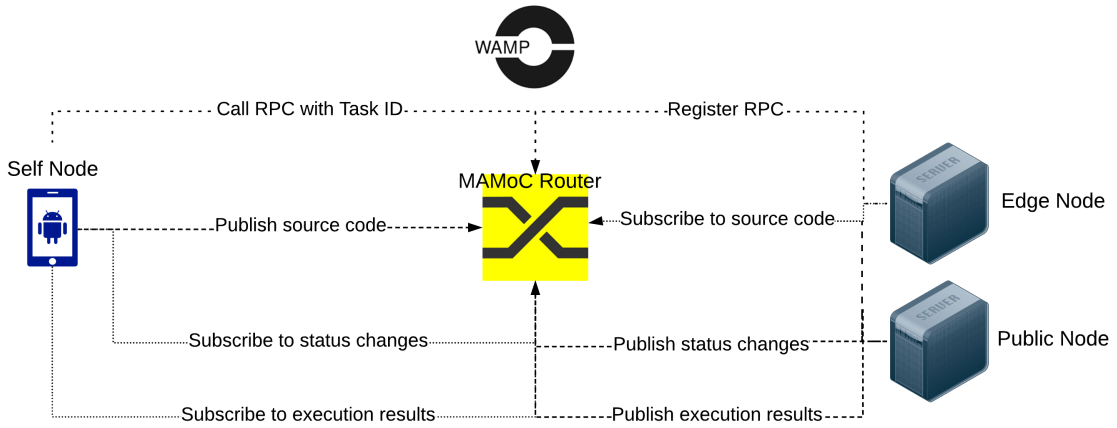


Fig. 5.4 RPC and Pub/Sub messages using WAMP

1. **RPC** MAMoC makes lightweight offloading decisions in the granularity of a particular task. It determines whether to run each individual task locally or on a remote site to optimise overall performance and energy consumption. As described earlier, MAMoC assumes that remote execution RPCs are segments of a program that were defined by the programmer to serve as candidates for remote execution, and are functionally idempotent and independent tasks. Similar to traditional asynchronous thread-safe RPCs, routed RPCs do not employ any shared memory and synchronization mechanisms and have no inter-thread dependencies. Each RPC event is launched in MAMoC Client as a separate thread.

   In MAMoC Client, the deployment controller constructs an RPC message and sends it over to the router component. The registration of the procedure in the server side is triggered in the first occurrence so that it can be called for future RPC events from the same Self Node or other Host Mobile Devices. The message flows between the Self Node and ENs and PNs are depicted in Figure 5.4.

2. **Publish/Subscribe**

   In MAMoC, SNs act as publishers for publishing the source code while they serve as subscribers for status changes and execution results. Meanwhile, ENs and PNs are subscribers for decompiled source code but publishers of server status changes and task execution results. Similarly, MAMoC router serves as the broker between the two sides.

   The existence of the task ID T_i^{ID} in MAMoC Server is a prerequisite for a smooth task offloading experience. Self Node must make sure that it can publish the source code to the server so that the code transformer component in the server manager can use it for the transformation process before being passed to the execution controller. The subscription to the event from server-side happens when no existing source code is found as depicted in Figure 5.1.

   Before sending any requests, Self Node registers the server with a unique ID in the service discovery process. To keep the status of the server up to date, the CPU information and existing container information is periodically published to the device profiler module. This information is updated in the helper database for aiding the subsequent offloading decisions. Self Node is also required to subscribe to the topic of the task execution after the RPC is

taken place. Since the execution is performed asynchronously, the execution result will eventually be published by the server to the mobile device.

5.4.3 Request Validation

The vulnerable nature and heterogeneity of wireless links have complicated the security and privacy needs in MCC [5]. Energy limitations in mobile devices extend the demand for a lightweight security mechanism. A fundamental part of such a security mechanism is authentication, which secures any system against unauthorised access. The most important security challenge for MCC applications is authentication. Authentication is bidirectional so the mobile device must authenticate to the service provider and the service provider must authenticate to the mobile device. Authentication should be lightweight to minimise computation and communication costs. Authorised access is the primary issue when the Host Mobile Device accesses the information stored on the cloud. Every access should be authenticated so that the mobile user can access information related to them only. No unauthorized access should be permitted, and this is accomplished by several authentication techniques such as providing login IDs, passwords, or PINs to the individual user to verify their identity which permits them to securely get access to their data.

In MAMoC, before being able to send offloading requests to offloading sites, Self Node needs to present credentials to the MAMoC router for authentication purposes. The ID and credentials of the router are previously set up between MAMoC Client and MAMoC Server to ensure a trusted execution environment and interaction. This approach is similar to the OpenID framework [113] which is a popular lightweight HTTP-based URL authentication. MAMoC router works as an identity provider and the server manager is the relying party for the mobile device user.

In securing D2D communications, Wi-Fi Direct requires all P2P devices to implement Wi-Fi Protected Setup (WPS) to secure the connection establishment process and communication in the P2P group. In WPS scheme, the GO implements the internal registrar whereas GC implements enrollee. The WPS scheme works in two phases [70]. First, the internal registrar generates and issues the network credentials to enrollee. Then, the enrollee (GC) reconnects to the internal registrar (GO) using the new credentials.

5.5 Task Execution Workflow

The four phases of MAMoC task execution workflow is illustrated in Figure 5.5. Subsection 5.5.1 describes the preparation phase, which involves the task annotation, code decompiling, and application refactoring processes. Subsection 5.5.2 is about offloading decision making, which was formerly represented in Chapter 4. The execution and post-execution phases in both Subsection 5.5.3 and Subsection 5.5.4 includes the steps of running the task either locally or remotely and analysing the execution results to assist subsequent offloading decision makings.

5.5.1 Preparation Phase

In order to support mobile applications to perform as required with MAMoC framework, some called for preparation steps need to be completed. Most of these arrangement techniques simply need to be handled during the initial launch of the mobile application. At this point, the application is reviewed whether it is manually annotated by the developers or it requires to be forwarded to the MAMoC Server to produce the annotations for the offloadable tasks. Another consideration for code offloading is to actually acquire the source code of the offloadable tasks from the application execution file for subsequently transferring them to the ENs and PNs.

5.5.1.1 Annotations

Application developers can use Java Pluggable Annotation Processing API to annotate the heavy tasks (in the level of classes or methods) with *@Offloadable* annotation. This can be regarded as metadata added to the Android source code and is attached to any class or method that can be offloaded to external candidate nodes without depending on any native library components in the mobile OS. An example class is the KMP [72] class, which solely depends on Java calls and can be executed on any node with a JVM interpreter. If the computation of the text search problem is given in a method that is invoked in an Android activity, then the method is annotated instead. The *@Offloadable* annotation interface has two boolean optionals:

- *parallelizable*: The tasks of the embarrassingly parallel programs can be passed independently. There are no dependencies between the subtasks of the task. An example is a text search task that can be split across a number of computing nodes with no data exchanges between them. The offloading node can partition the task (the text file in this instance) and send it over

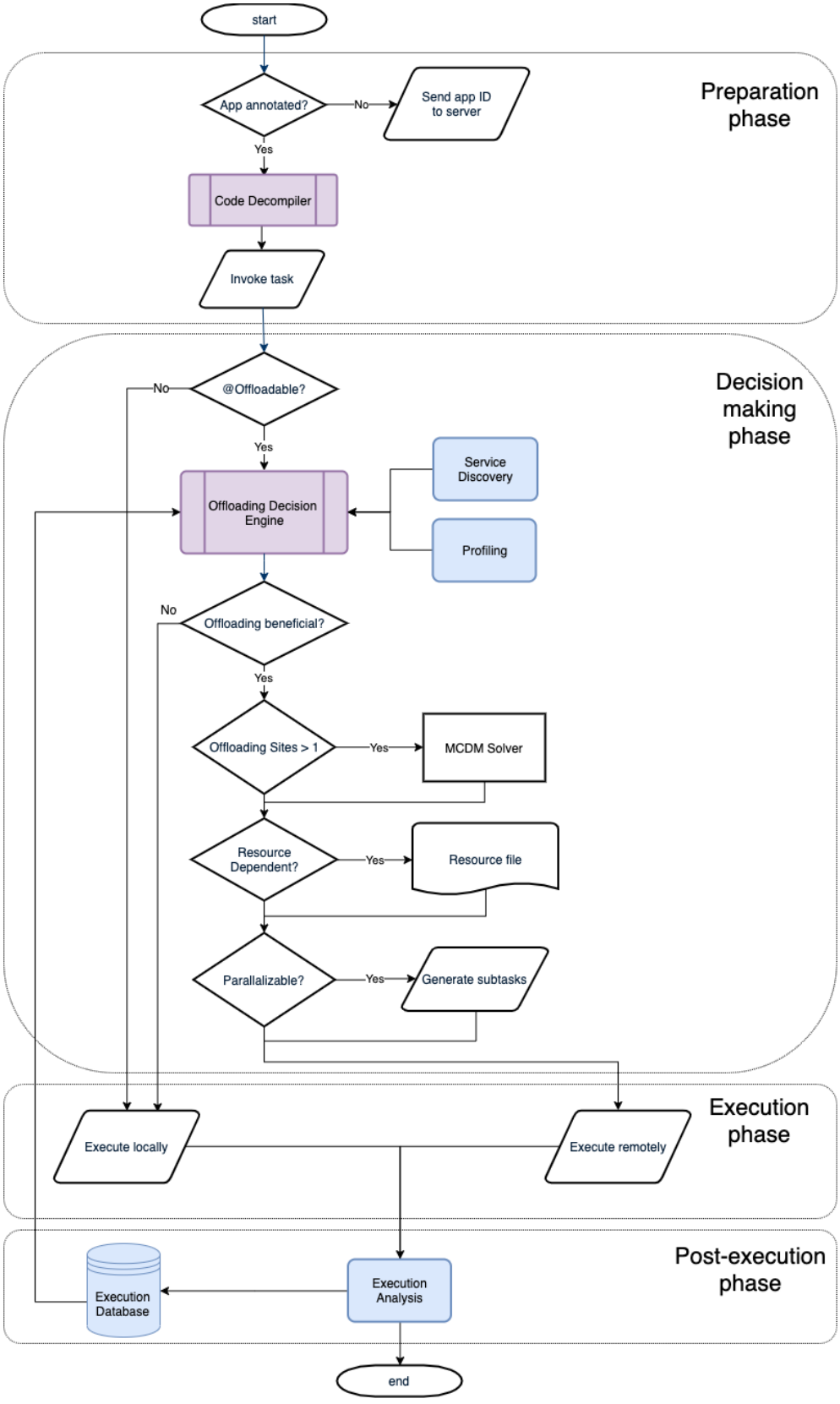


Fig. 5.5 MAMoC task execution workflow

to external resources. After the results are returned, they are merged and presented as a single result in the same sense as if the execution were to take place locally.

- *resourceDependent*: The tasks dependent on resources need to be accessible at the time of processing. Examples of resources needed in mobile apps can be in the form of text files (word search and sorting workloads in word processing apps), images (face detection and recognition apps), audio files (translation apps). The resource files in Android apps are statically added to the Assets folder or the assigned resources directory, which includes XML files for layout design and global values. Any *@Offloadable* class or method, which has set this optional element to true, requires the data to be present at the remote site before being processed.

```
@IndexAnnotated
public @interface Offloadable {
    boolean parallelizable() default false;
    boolean resourceDependent() default false;
}
```

Listing 5.1 Java @Offloadable annotation used for offlodable tasks on MAMoC

5.5.1.2 Code Decompiling

Android developers upload an APK, which comprises the program bytecode, resources and an XML manifest, to the Google Play Console [3]. The submitted applications are originally developed in Java or Kotlin, but compiled into Dalvik bytecode and coalesced into a *.dex* file [102]. The Android devices incorporate a Dalvik Virtual Machine (DVM) which is different from conventional JVM. To retrieve the original Java source code composed by application developers, researchers and practitioners have developed reverse engineering tools such as Jadx[4], Apktool[5], and JEB[6].

A source-level API is written on top of Jadx core files in the MAMoC Client [7] for decompiling the dex file into Java source code and gaining resource files if desired. It functions by decompiling the classes.dex (sometimes classes2.dex too

[3]https://play.google.com/apps/publish/
[4]https://github.com/skylot/jadx
[5]https://ibotpeaches.github.io/Apktool/
[6]https://pnfsoftware.com/
[7]https://github.com/mamoc-repos/MAMoC-Client/tree/master/mamoc_client/src/main/java/jadx

for bigger apps) to an intermediary language called SMALI[8], which is akin to ASM[9] language, used to represent the Dalvik VM opcodes as a human-readable language and then converting the SMALI back to Java code.

```
private void decompileAnnotatedClassFiles() {
    ArrayList<String> classNames = new ArrayList<>();

    for (String s :ClassIndex.getAnnotatedNames(Offloadable.class)){
        classNames.add(s);
    }

    DexDecompiler decompiler = new DexDecompiler(mContext,
        classNames);
    decompiler.start();
}
```

Listing 5.2 Java code for decompiling the annotated classes

5.5.1.3 Application Refactoring

In the event of mobile applications on the Play Store that are not manually annotated, the application execution file needs to be examined and partitioned into offloadable and non-offloadable tasks. This procedure is performed in the MAMoC Server by first loading the APK file of the application, decompiling the file to SMALI code, running a static analysis to determine the heavy tasks, adding needed annotations to the offloadable tasks, and eventually recompiling the decompiled code into a signed APK to be installed on the mobile devices. A copy of the decompiled Java source codes for the offloadable tasks are further stored to the MAMoC Repository for future offloading request calls. Section 5.7.2.4 yields the implementation details of this process.

5.5.2 Decision Making Phase

Chapter 4 provided an overview of the proposed modelling and algorithms for task offloading. When the offloadable task is invoked, a decision needs to be carried out at runtime, whether the task should be offloaded and where to be offloaded. The offloading decision engine uses profiling and past execution records to deal with those queries. The offloadable tasks might depend on resource files and can be run independently in a parallel fashion. Both conditions need to be dealt with in the decision-making phase.

[8]https://github.com/JesusFreke/smali/wiki
[9]https://asm.ow2.io/

5.5.2.1 Profiling

- *Network profiler:* network communication constitutes one of the largest sources of energy consumption in a mobile application [107]. According to an energy profiling study, network communication consumes between 10-50% of the overall energy budget of an ordinary mobile application [106]. A network profiler collects information about wireless connection status and available bandwidth. This profiler runs asynchronously at runtime so it can record any shifts in the network such as the state, signal strength, and bandwidth of Wi-Fi or cellular connections. Because of the mobile nature, network status may vary periodically (e.g. user moves to different locations). It also calculates the Round Trip Time (RTT) to the nodes once a successful connection is established. It measures the throughput by sending packets to the connected nodes, which eventually estimates the device's upload rate [61]. The node then sends back packets that enable it to resolve the device's download rate. Updated information about a wireless link is vital for the decision engine to produce correct offloading decisions [76].

- *Device profiler:* the hardware operating conditions of the mobile device are gathered and analysed. This profiler monitors and collects hardware context data asynchronously at runtime. From the Host Mobile Device, it picks up the hardware values including the device's total and available RAM, the number of CPU cores, and the average and maximum clock of each core. It also monitors the battery state (charging, not charging) and battery level (0-100) of the Self Node and MNs.

5.5.3 Execution Phase

The outcome of the task offloading decision making is either local or remote execution. In the event of local execution, the task is performed on the Host Mobile Device but since the offloadable tasks still need to be moved to the framework for decision-making purposes; The framework needs to supervise the execution. Remote execution can appear in two divergent forms on either Nearby Mobile Device or Cloudlets and Remote Clouds.

5.5.3.1 Local Execution

This form of execution comprises all the unoffloadable tasks that should be unconditionally executed on the Host Mobile Device and the offloadable tasks. The unoffloadable tasks have codes that access local device units such as the

camera, GPS, accelerometer, or other sensors. No offloading decisions must be taken for these tasks. However, as for the offloadable entities, the decision engine which was illustrated previously may execute the task locally if the Performance Gain PG or Energy Gain EG are less than zero.

5.5.3.2 Remote Execution

According to the outcome of the task offloading engine, the task or its parallelized subtasks can be executed on either mobile, edge or public nodes. The remote execution mode is different for the mobile node than to the edge and public nodes because of their architecture differences.

- *Mobile Node* it is assumed that the mobile application is already installed on MNs which are also running on Android OS. MAMoC Client uses Java Reflect API which allows inspection of classes, interfaces, fields and methods at runtime without the need of knowing them at compile time. Since MAMoC sends over the task (class or method) ID. The Mobile Node will execute it by instantiating a new object and invocation of the class or invoking the method.

- *Edge Node and Public Node:* traditional MCC frameworks deploy the same mobile runtime environment in the form of VMs in the cloud to support offloading [31] [69] [73]. This process makes it straightforward to run mobile code on the cloud side. However, to use a VM instance, the cloud has to install and boot a guest as in the VM, which incurs a substantial delay. Heavyweight virtual machines deployment for mobile cloud offloading requires additional resources on the computing host [127]. OS-level virtualization called *containers* usually imposes less overhead than VMs, since they share the host with the kernel and do not suffer the overhead of resource virtualization. MAMoC uses the concept of right-size containers [80] which is a container that is created on the offloading site for the offloaded code is of the smallest size possible to run the offloaded computation, based on computation requirements metadata related to the offloaded code, in order to optimise resource usage on the site.

  After receiving the task ID (ID_{T_i}) from Self Node, the site first checks if the computation already exists. For the first time, the site cannot locate the computation internally. Hence, Self Node sends the computation over to be registered for future procedure calls. The Publish mechanism of Pub/Sub is used to publish the decompiled source code of the task. After the task

is executed successfully, the execution result is sent back to Self Node and the task ID will be registered in the server. For future executions, only the invocation parameters need to be received from SNs.

The server components can be started once and left running to wait for mobile client requests. This mechanism is used in [73] where a startup process starts all the services. Another approach is to wait for the offloading request to arrive before starting the offloading server as it is used in [123]. The running services get destroyed once the offloading request is served and the mobile device terminates the connection. For the latter approach, the waiting time of the offloading request increases and affects the user experience of the mobile user because of the long server setup delay time. However, it can be useful for scalability and elasticity by following the just-in-time approach [54]. MAMoC Server keeps running as detached Docker containers in the service providers.

5.5.4 Post-execution Phase

After the execution is completed locally or remotely, an analysis is planned and the execution details are registered in a database for assisting in prospective decision making of task offloading.

5.5.4.1 Execution Analysis

There is a demand to inspect and interpret execution results in the post-execution analysis phase to classify data outputs (e.g. whether the execution result make sense), examine execution traces and data dependencies (the results that were corrupted by this input dataset), debug runs (The reason a step fails), or purely analyse performance (Finding the steps that took the longest time). The execution analysis entries are immensely dynamic, with the newest measured records receiving high priority, while the oldest ones are discarded after a while. During the implementation phase, MAMoC performs comprehensive experiments to determine the number of entries per task execution to hold in the database.

This analysis is primarily performed for updating the values of Maximum Local Executions ($MaxLE$) and Maximum Remote Executions ($MaxRE$) which are checkpoint variables in the task offloading decision algorithm as illustrated in Algorithm 4.1. Initially, the values of checkpoint variables ($MaxLE$ and $MaxRE$) are statically set to five. It is necessary to introduce some learning mechanisms to update those values dynamically. MAMoC uses a simple heuristic to increment the value by one every time the other execution is less favourable than current

execution, e.g. incrementing $MaxRE$ by one when remote execution is proven to be more beneficial than local execution.

5.5.4.2 Helper Database

This is employed to keep a history of all the previous task executions for the decision engine to utilise it as a supplementary element when the location of task offloading is decided upon. The local datastore is managed via SQLite, an embedded SQL database. The SQLite is chosen due to its popularity, light-weightiness and versatility, which makes it the most widely deployed and widespread SQL database for mobile development. Even though other database solutions such as CoucheBase, MongoDB, and Cassandra have a more flexible schema and built-in synchronisation [42], MAMoC does not require these extra features rather than the past local and remote executions which are later adopted in the decision-making process.

The database fields are presented in both Table 5.1 and Table 5.2 collectively with a brief description. Once a task is executed, the costs consisting of the execution location, processing time, and energy consumption are recorded in the database. The stems of analysis are kept in the profile database for the offloading policy manager to create a dynamic offloading decision on behalf of the user.

Field name	**Description**
IpAddress	The IP address of the node
CpuFreq	The CPU frequency in MIPS
CpuNum	Number of CPU cores
Memory	Memory size in Bytes
JoinedDate	The timestamp of establishing connection with the node
BatteryLevel	Battery level between 1-100
BatteryState	True if charging, False otherwise
OffloadingScore	Calculated offloading score of the node

Table 5.1 Node table and its fields with their description

Field name	**Description**
ExecutionID	A unique ID of task execution
AppID	The unique Android application ID that looks like a Java package name, such as com.example.myapp. It uniquely identifies the app on the device and in Play Store.
TaskID	It consists of the app ID, package name, and class or method name, such as uk.ac.standrews.mamoc_demo.textsearch.kmp
ExecutionLocation	It is where the task was executed. It can be LOCAL, MOBILE, EDGE, or PUBLIC
NetworkType	Can be one of the following types: MOBILE, WIFI, WIMAX, ETHERNET, and BLUETOOTH. Obtained using the Android Network Info API
ExecutionTime	Time in nanoseconds (ns) it took to execute the task locally or remotely. In case of offloading this covers the time spent on data transmission and the time spent on collecting the result.
CommOverhead	Time in nanoseconds (ns) it took to transmit source code or resource files.
RttSpeed	The RTT between the device and the offloading site. Measured during the offloading process.
ExecutionDate	The timestamp in milliseconds when the task was executed. The entries of the database are sorted in decreasing order based on this field, to give priority to the most recent executions.
Completed	True if task was executed successfully, False otherwise.

Table 5.2 ExecutionHistory table and its fields with their description

5.6 MAMoC Client

MAMoC Client and programming APIs are built as an Android external library using Java programming language for Android application developers. For the sake of easy integration, the library is distributed[10] to Bintray JCenter [11] which is a popular public repository for open source libraries. Gradle[12] provides built-in shortcut methods for the most widely used repositories, including jcenter(). The build file of Gradle plugin in Android Studio can then simply include the library as a dependency. Appendix 5.8 provides a longer hands-on tutorial on the integration of the framework to existing Android projects.

5.6.1 Service Discovery

The service discovery module is initiated by the MAMoC framework once the application is opened on Self Node. It first connects to the router module in MAMoC Server to access all the available server manager modules that can execute the offloadable tasks. Meanwhile, it starts a new thread in the background, which advertises the service to let the Nearby Mobile Devices of the service and also discovers the available services as described earlier in Section 5.4.

Notably, the Edge Node can be accessed in two modes:

- Using the Avahi daemon installed on the ENs which advertises the local service of the Edge Node to the mobile devices over the LAN. In this mode, the ENs are treated as nearby devices but due to architectural differences with the Nearby Mobile Device mobile devices, they are still treated as MAMoC Server service providers.
- Using a pre-allocated IP address of a local router component that is deployed in a nearby offloading site. This mode works the same way as connecting with the PNs over the WAN as depicted in Figure 5.6.

A standard user interface is developed to be used by the applications for service discovery and managing device connections, as shown in Figure 5.6.

[10]https://bintray.com/dawand/mamoc_client
[11]https://bintray.com/bintray/jcenter
[12]https://gradle.org/

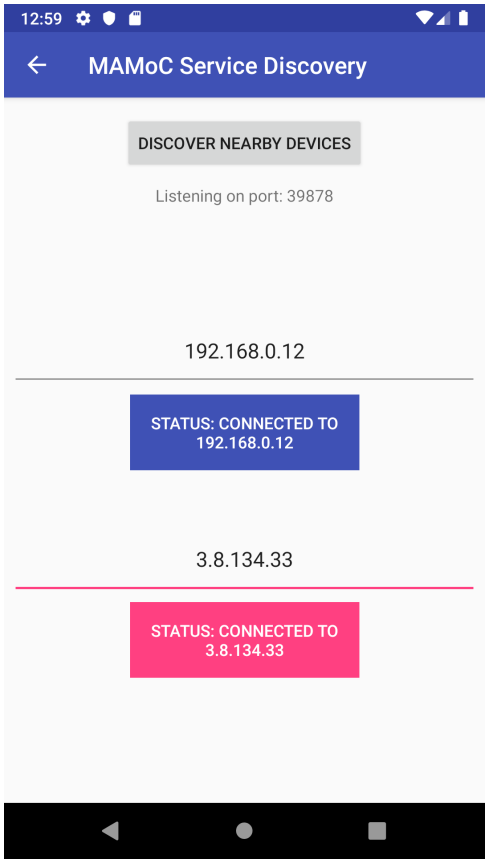


Fig. 5.6 Service Discovery Android Activity

5.6.1.1 Device to Device

Android provides helper libraries for managing D2D communications. As described earlier in Section 5.4, the communication with Nearby Mobile Devices can be done using infrastructural Wi-Fi, where the devices are connected to the same AP using Network Service Discovery (NSD) or Wi-Fi P2P for direct D2D communications without being connected to a network. These libraries handle different P2P computing issues like peer naming, peer and resource discovery, resource-metadata handling, message routing and resource delivery. It thus provides an infrastructure to deploy mobile P2P applications over ad-hoc networks.

The implementation details of both techniques are described below:

- **Network Service Discovery (NSD):** This connection is used when the devices are connected to the same wireless AP (a.k.a router or WiFi hot spot). Android NSD allows the Host Mobile Device to identify devices in the same network that offer the requested services. With NSD, it is possible to register, discover and connect with the intended service (or other services) over the network.

```
public class WiFiP2PSDActivity extends AppCompatActivity implements
    WifiP2pManager.ConnectionInfoListener {

    private static final String SERVICE_INSTANCE = "MAMoC";
    private static final String SERVICE_TYPE = "_http._tcp";

    private MobileNode selfNode;

    private void initialize() {
        selfNode = MamocFramework.getInstance(this).getSelfNode();
        ServiceDiscovery serviceDiscovery = ServiceDiscovery.
            getInstance(this);
        serviceDiscovery.startConnectionListener();

        wifiP2pManager = (WifiP2pManager) getSystemService(
            WIFI_P2P_SERVICE);
        wifip2pChannel = wifiP2pManager.initialize(this,
            getMainLooper(), null);

        startRegistrationAndDiscovery(Utils.getPort(this));
    }
}
```

Listing 5.3 WiFiP2P service initialization

The iOS equivalent of Android's NSD is called *MultipeerConnectivity*[13] library. I used this library for the process of discovering nearby devices running MacOS (MacBook, iMac) and iOS (iPhone, iPad) which are connected to the same local network [135]. Similarly, the library uses the concept of zero configuration network technology [30] which enables devices to advertise services and to discover what services other nearby devices on the local network are offering.

- **WiFi P2P (Wi-Fi service discovery):** This mode does not need an AP to be in the Wi-Fi range. If the Host Mobile Device is a GO, a list IP addresses of the Nearby Mobile Device is fetched. Otherwise, the IP addresses are resolved from GO. Every peer in the group should know the IP address of GO. The remaining problem is sharing the port it is listening on. We can prefix the port, then sharing the dynamic port data with clients, and then clients can share their info and dynamic port. So first communication happens over the fixed port and after that, the dynamic port data number is transferred before the regular socket communication.

  Wi-Fi Direct is a relatively newer technology which was introduced from the Android 4.0 version. The Android Wi-Fi Direct interface (WifiP2pManager)

[13] urlhttps://developer.apple.com/documentation/multipeerconnectivity

[14] allows developers to discover, request, and connect to peers and provides listener methods that detect the success or failure of connected and dropped connections as well as newly discovered peers. The WifiP2pManager API is asynchronous and responses to requests from an application are on listener callbacks provided by the application. The application needs to do an initialization with *initialize(Context, Looper, WifiP2pManager.ChannelListener)* before doing any P2P operation as depicted in Listing 5.3. Similar to NSD, the service can be registered and discovered over Wi-Fi direct. For that, we can use a method of WifiP2pmanager called addLocalService() as depicted in Listing 5.4.

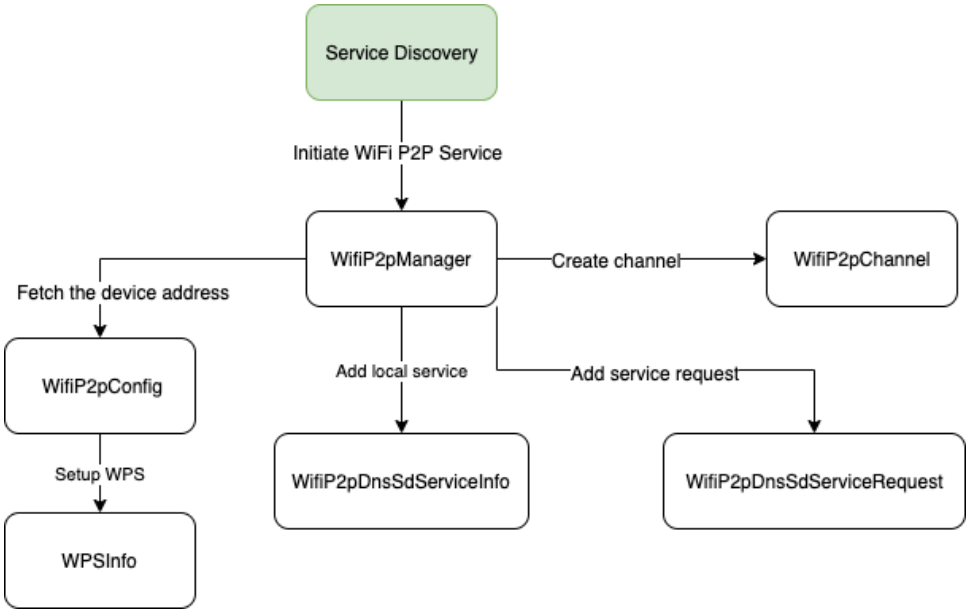


Fig. 5.7 WiFi P2P Service Discovery Process

The service calls need an *ActionListener* instance for receiving callbacks ActionListener#onSuccess or ActionListener#onFailure for updating the list of discovered MNs. Action callbacks indicate whether the initiation of the action was a success or a failure. Upon failure, the reason of failure can be one of *ERROR*, *P2P_UNSUPPORTED* or *BUSY*.

Notably, this type of communication does not use broadcasting techniques as it works in P2P fashion. Using WifiP2pDnsSdServiceInfo, there is no connection setup as compared to zero configuration and multicast broadcast talk/listen. With a broadcast communication, the battery drains faster with sending constant broadcast messages. With the WiFi P2P connection, once

[14]https://developer.android.com/guide/topics/connectivity/wifip2p

```
private void startRegistrationAndDiscovery(int port) {

    Map<String, String> record = new HashMap<String, String>();
    record.put("DeviceID", selfNode.getDeviceID());
    record.put("LocalIP", selfNode.getIp());
    record.put("PortNumber", String.valueOf(port));
    record.put("DeviceStatus", "available");

    WifiP2pDnsSdServiceInfo service = WifiP2pDnsSdServiceInfo.
        newInstance(SERVICE_INSTANCE, SERVICE_TYPE, record);
    wifiP2pManager.addLocalService(wifip2pChannel, service, new
        WifiP2pManager.ActionListener(){...});

    discoverService();
}
```

Listing 5.4 Registering local WiFiP2P service

the service is discovered, and the connection is established, it will work at a similar speed/power-efficient as regular Wi-Fi connections.

5.6.1.2 Device to Server

The communication between the Host Mobile Devices and ENs and PNs service providers is implemented using WebSocket over TCP. Listing 5.5 depicts the interface that the Edge Node and Public Node implement in order to establish the full-duplex WebSocket connections and send messages in both directions. MAMoC Client hosts a client library that can communicate with the MAMoC router module and send RPC and Pub/Sub event messages. This will be explained in more detail in Subsection 5.7.1

```
public interface IWebSocket {
    void connect(String wsUri, IWebSocketConnectionHandler wsHandler
        ) throws WebSocketException;
    boolean isConnected();
    void sendMessage(String payload);
    void sendMessage(byte[] payload, boolean isBinary);
    void sendPing();
    void sendPong();
}
```

Listing 5.5 The WebSocket interface for managing the edge and public node connections

5.6.2 Code Decompiler

This section describes the process of annotating the heavy tasks to make them accessible to MAMoC Client devices to perform the offloading process. Since the offloading can be done in class and method levels, the annotation can be added to the whole class or a particular method within a class. First, Section 5.6.2.1 investigates the process of annotation indexing and explains the features of the ClassIndex library that is adopted in MAMoC. In the case of no manual annotations, the offloadable snippets need to be identified from an app binary.

5.6.2.1 Annotation Indexing

As described in Section 5.5.1.1, the heavy tasks of MAMoC-enabled mobile applications are annotated with @Offloadable annotation. The annotation has two boolean optionals of whether the task needs any data to be present at the remote site before being processed and whether the task is parallelizable.

Traditional classpath scanning is a slow process. Instead, MAMoC replaces it with compile-time indexing that speeds Java applications bootstrap considerably. ClassIndex uses annotation processing to generate the index of classes at compile-time and put it with the rest of the compiled class files. The index is then available at run-time using the getResource() method which must be implemented by all classloaders. Section 5.6.2.2 explains the process of decompiling the bytecode instructions to Java source code to be transferred to the MAMoC Server for performing code transformation and executing it. Table 5.3 shows the results of the benchmark comparing ClassIndex with various scanning solutions.

Table 5.3 Annotation indexing library comparisons conducted in [9]

Library	**Application startup time**
Scannotation [15]	0:05.11
Reflections [16]	0:05.37
Reflections Maven Plugin [17]	0:00.52
ClassIndex [9]	0:00.18

Complexity is reduced by annotating classes and methods for possible remote execution instead of preparing separate versions of code for the mobile device and the server. The *@Offloadable* annotation is marked with @IndexAnnotated meta-annotation. This creates at compile-time an index file in

[15]https://mvnrepository.com/artifact/org.scannotation/scannotation/1.0.3
[16]https://github.com/ronmamo/reflections
[17]https://github.com/ronmamo/reflections-maven

```
private void decompileDexFile(File dexInputFile) {
    JadxDecompiler jadx = new JadxDecompiler();
    jadx.setOutputDir(getOutputDir(context));
    jadx.loadFile(dexInputFile);
    jadx.saveAnnotatedClassSources(classNames);
}
```

Listing 5.6 decompileDex method that converts a .dex file to Java source code

META-INF/annotations/uk/ac/standrews/mamoc_client/Offloadable directory, which contains all the annotated classes. The index file can then be accessed by MAMoC Client at runtime to find the offloadable classes without the need to scan the classpath.

5.6.2.2 Dex Decompiler

In a traditional Java application, all the Java source files are executed into .class files, which consist of bytecode instructions to be executed on Java Virtual Machine (JVM). Since mobile devices are severely constrained in the amount of memory, processing power, and battery life available, the Android Runtime (ART) is used which is a more lightweight runtime compared to JVM [33]. ART offers superior performance to the JVM by performing both Ahead-of-Time and Just-in-Time compilation. However, ART uses incompatible opcodes, so an additional step is required, where .class files are converted into a single *classes.dex* file. The Dalvik Executable format (DEX)[18] file references all the classes or methods used within an app. All the components of an Android application such as Object, Activity, and Fragment will be transformed into bytes within a Dex file that can be run as an Android app.

The DEX format has a limit to the number of classes that can be declared in a single dex file, so it is essential to check for the existence of *classes2.dex* file as well. At some point, Android apps became bigger so Google had to adapt this format, supporting a secondary .dex file where other classes can be declared. The implemented tools need to be able to detect the multi-dex files and append them to the decompilation pipeline.

MAMoC uses Jadx [19] library to decompile the .dex file(s) to Java source code in the Android application as shown in Listing 5.6. It provides a clean and easy-to-use API file to be included in the framework for performing the decompilation process as depicted in Figure 5.8.

[18]https://source.android.com/devices/tech/dalvik/dex-format.html
[19]https://github.com/skylot/jadx

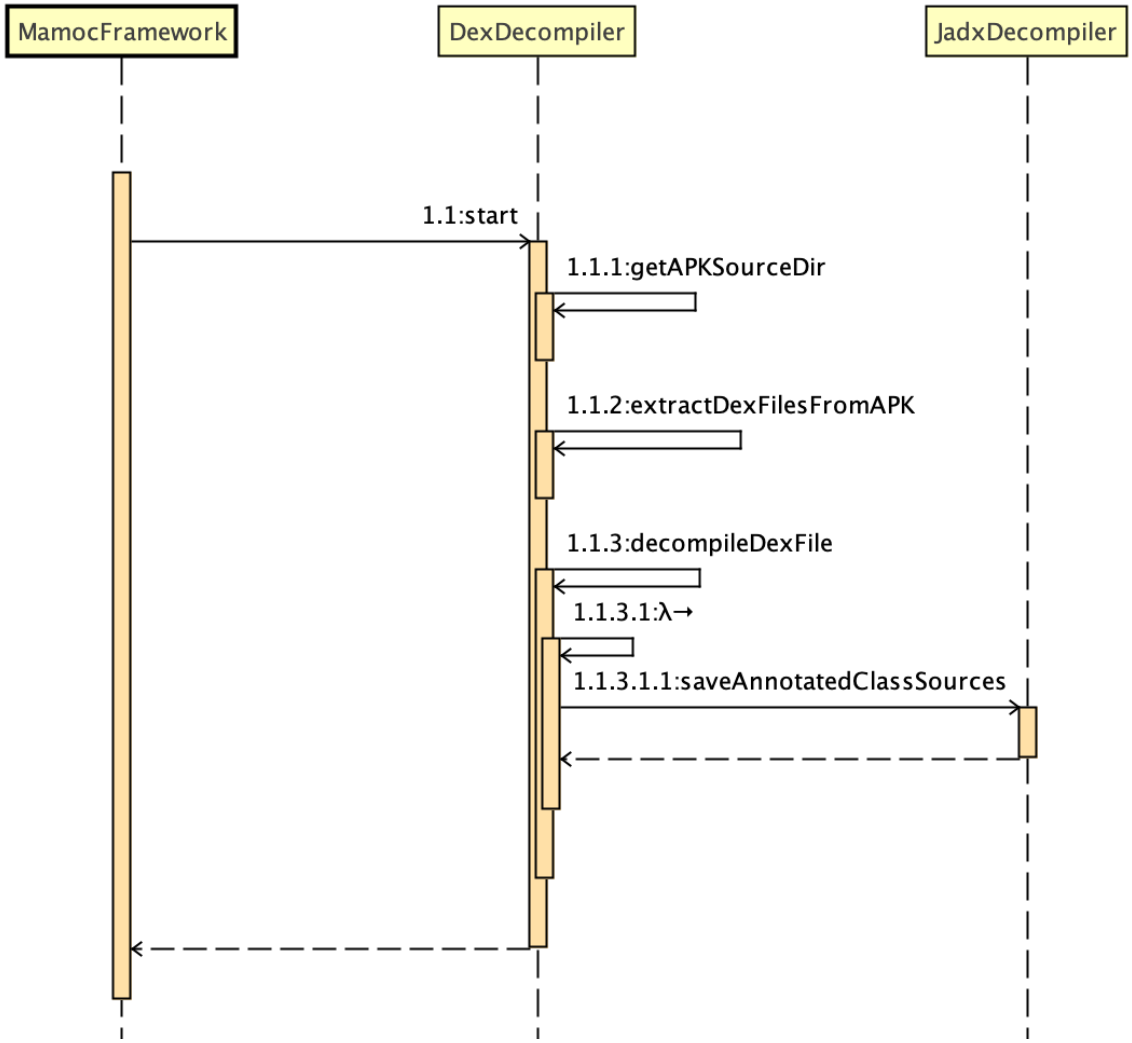


Fig. 5.8 Dex decompiler Sequence Diagram

5.6.3 Context Profilers

The hardware figures gathered by the device profiler illustrate the running conditions of the observed mobile device. The device profiler collected information is supplied to the decision engine when required to aid the offloading decision making. This profile consists of collecting the total CPU frequency, average CPU frequency, the number of CPU cores, the total and available memory, and the battery level and charging state. It reads native system files for getting capability and usage from "/proc/stat" and "/sys/devices/system/cpu/..." for each CPU core. The battery profiler uses a combination of Android API and reading native system files from Android's BatteryManager API in "/sys/class/power_supply".

The network profiler collects the network status of the mobile device asynchronously at runtime so that it can report any changes in the background. The

following network conditions are monitored: cell connection state and its bandwidth, WiFi connection state and its bandwidth, the Round Trip Time (RTT) to the connected nodes (congestion level of each connection), and the signal strength of cell and WiFi connection. *WifiManager* class is used for detecting the link speed of the Wi-Fi connection while *TelephonyManager* is employed for detecting the type and speed of the cellular connection (3G or 4G). ConnectivityManager[20] Android library class interrogates the state of the network connectivity to alert applications when network connectivity changes.

In order to avoid the overheads of polling, *BroadcastReceiver* class is used which has an *onReceive* method that receives the Intent containing either the battery level and state changes or the network state tracking Intent that contains the active network information. This helps MAMoC monitor changing conditions only when the network state or the battery level or state has actually changed.

5.6.4 Offloading Decision Engine

The decision engine is the most crucial component in MAMoC Client since an imprecise decision will lead to extra resource expenditure and wasted battery consumption. The decision making is highly dependent on the inputs of the other components including service discovery, context profilers, and database adapter.

The offloading decision making process was modelled and explained in detail in Chapter 4. As depicted in Figure 5.9 generated using an Android Studio plugin [121], the decision engine first fetches the connected nodes from the service discovery in a *TreeSet* to avoid fetching duplicates through rediscovering them and keep the order of the nodes from highest to lowest according to their profiling information. It then fetches the device and network profiling information of the available nodes to calculate their offloading scores for deciding whether it is worth offloading. The next step is to fetch the past local and remote execution records for the invoked task and if more than a node is available for multisite offloading, multi-criteria decision methods of AHP and fuzzy TOPSIS are employed for evaluating and ranking the nodes.

[20]https://developer.android.com/reference/android/net/ConnectivityManager.html

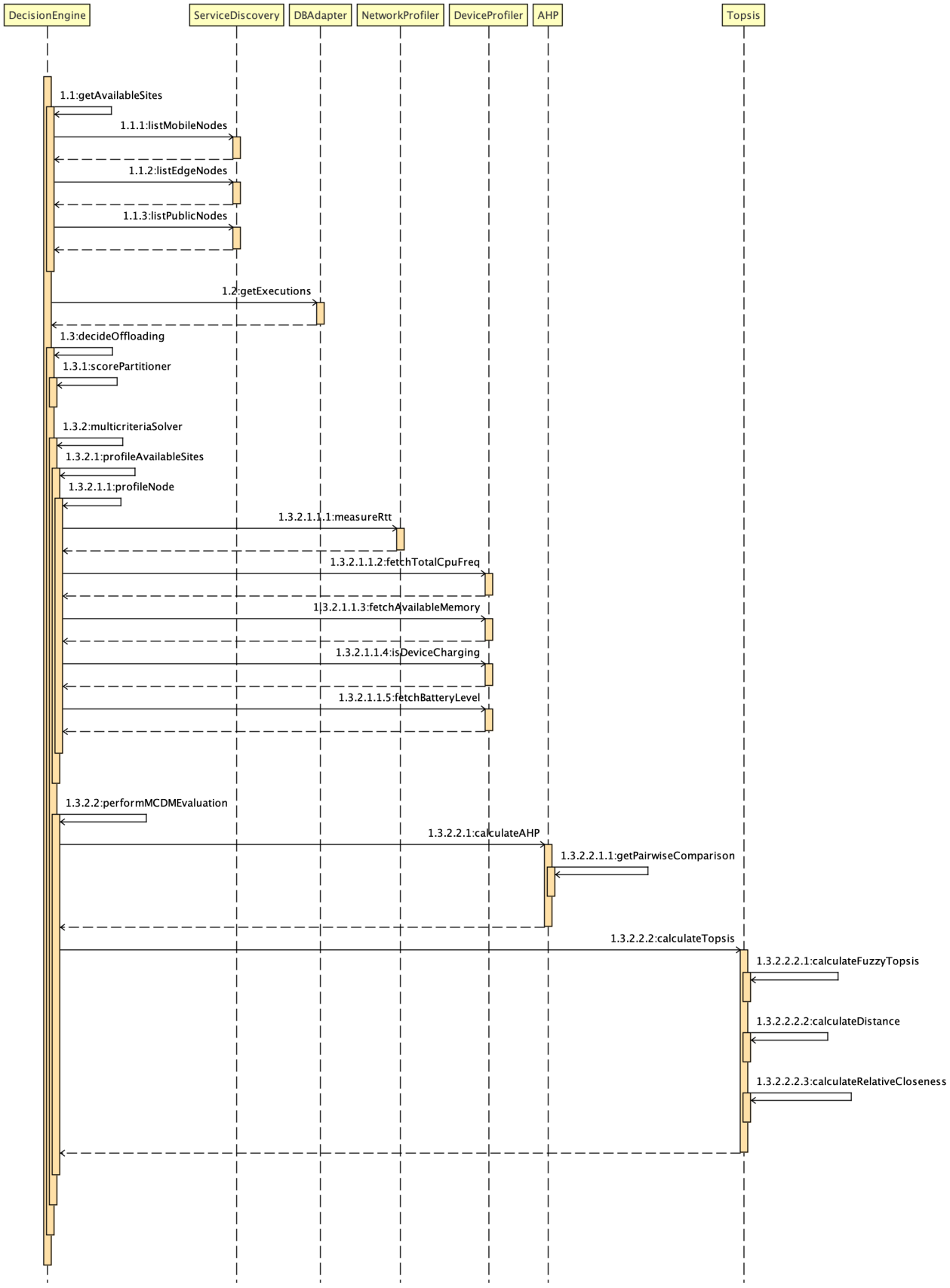


Fig. 5.9 Decision Engine Sequence Diagram

5.6.5 Deployment Controller

The deployment controller is responsible for communicating with the connected nodes and manage the remote execution by calling the remote endpoints or publishing the decompiled source codes and finally receiving the execution results back. Decision engine either decides to run the task locally or passes the node with the highest offloading score in the case of single-site offloading or passes the list of offloading sites and their offloading percentages generated from the MCDM evaluation in the case of multisite offloading.

Based on the decided execution location(s) from Algorithm 4.1, the following scenarios occur in order to proceed with the local and remote task executions:

1. **Remote execution (single-site offloading)**

    - If the node is a Mobile Node: as mentioned before in Section 5.2, it is assumed that the Nearby Mobile Devices are already equipped with the MAMoC-enabled mobile applications. When an offloadable task is to be executed, the deployment controller in the Self Node utilises Java Reflect to dynamically invoke the method **run** of the offloaded class instance with the provided parameters. Since the resources are already bundled with the installed application, there is no need to transfer them in the pre-execution phase. However, if the task has a dependency on a resource file, we need to access the appropriate constructor if there are any necessary parameters passed to it.
    - If the node is a Edge Node or a Public Node: the assumption is that the task ID (ID_{T_i}) of the offloadable task is registered in the offloading site (by either publishing the source code in the preparation phase described in **??** or during the application bytecode analysis in the server which will be explained later in Section 5.7.2.4). The deployment controller places an RPC call with the ID_{T_i} and the necessary parameters of the task as shown in Figure 5.10.

2. **Remote execution (multisite offloading)**

    - As shown in Algorithm 4.4, the multi-criteria solver generates a dictionary of execution locations and their offloading percentages. The deployment controller uses this information to divide the task into a number of subtasks equal to the number of nodes in the dictionary.
    - If one of the nodes is a Mobile Node, the class and method name invocations are similarly performed with Java Reflect but the constructor

parameters are going to be set according to the offloading percentage of that particular Mobile Node.

- If one of the nodes is an Edge Node or a Public Node, the same number of RPC calls as the subtasks will be fired in a parallel fashion and subscribe to their results.
- The returned subtask executions need to be collected and merged into a single execution result as it has happened as a single execution.

3. **Local Execution**

- Instead of returning the execution of the program to the calling point in the mobile application, MAMoC performs the local execution through Java Reflect similar to the remote execution on the MNs. The main goal of doing this is to monitor the execution environment and store the execution results in terms of execution time and energy consumption of the mobile device. The local execution sequence diagram is shown in Figure 5.11.

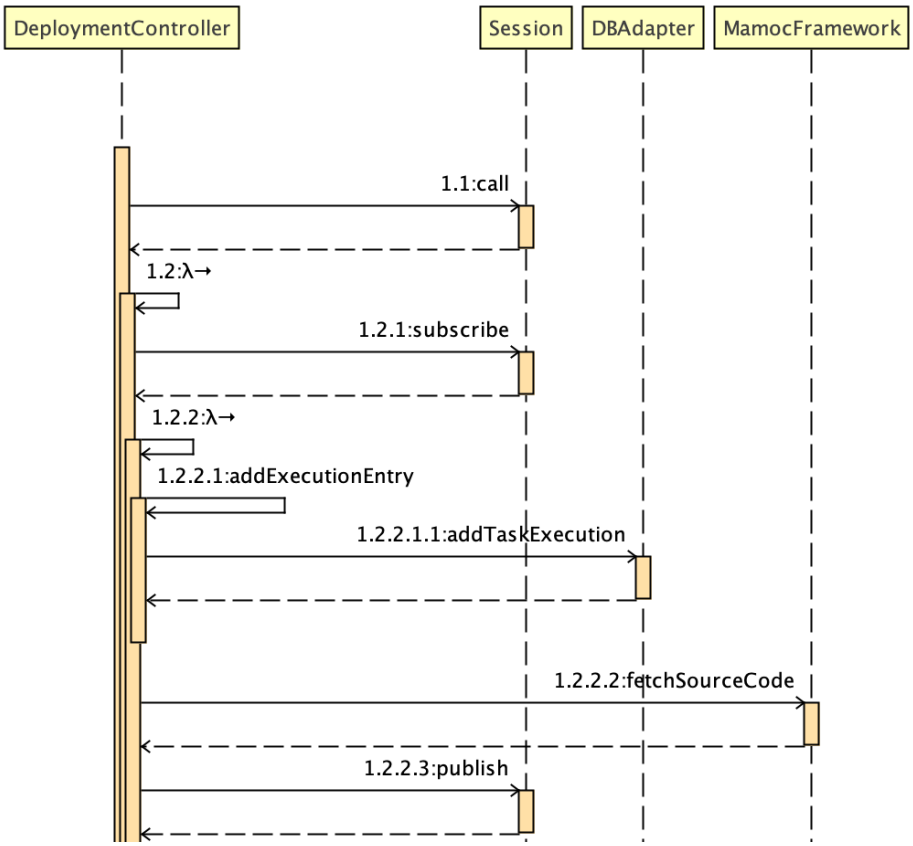


Fig. 5.10 Remote execution using RPC and PubSub sequence diagram

After the execution is performed successfully in the Self Node or the offloading sites, *LocalBroadcastManager*[21] which is a support package singleton class in Android library tools is used. This class is an application-wide event bus that embraces layer violations in the application to enable exchanging events from different components. The activity which expects execution results and duration from the framework needs to implement a *BroadcastReceiver* and register itself to the *LocalBroadcastManager* instance. The framework will then use the *sendBroadcast* method to broadcast the results of the execution, makespan, and communication overheads in the case of remote offloading.

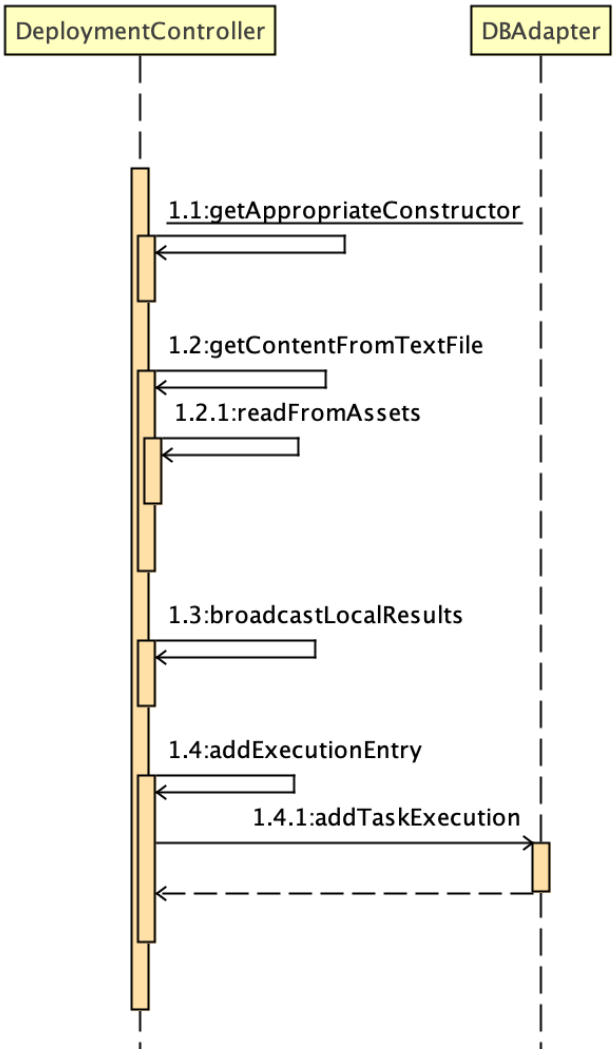


Fig. 5.11 Local execution on the host and nearby devices sequence diagram

[21]https://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager

```
public ArrayList<TaskExecution> getExecutions(String taskName,
    boolean Remote){

    ArrayList<TaskExecution> taskExecutions = new ArrayList<>();
    Cursor cursor = db.rawQuery("select * from " + TABLE_OFFLOAD,
        null);

    int id = cursor.getColumnIndex(TASK_ID);
    int location = cursor.getColumnIndex(EXEC_LOCATION);
    int networkType = cursor.getColumnIndex(NETWORK_TYPE);
    int execTime = cursor.getColumnIndex(EXECUTION_TIME);
    int comm = cursor.getColumnIndex(COMMUNICATION_OVERHEAD);
    int rtt = cursor.getColumnIndex(RTT_SPEED);
    int offDate = cursor.getColumnIndex(OFFLOAD_DATE);
    int completed = cursor.getColumnIndex(OFFLOAD_COMPLETE);

    taskExecutions.add(...); // add the fetched column indexes to
        the arraylist
    return taskExecutions;
}
```

Listing 5.7 getExecutions method in the client database adapter

```
ArrayList<TaskExecution> remoteTaskExecutions = framework.dbAdapter.
    getExecutions(taskName, true);

ArrayList<TaskExecution> localTaskExecutions = framework.dbAdapter.
    getExecutions(taskName, false);
```

Listing 5.8 Fetching local and remote task executions from the database in the offloading decision engine module

5.6.6 Database Adapter

The helper database contains a history of all the executed offloadable tasks that were performed locally or remotely. The deployment controller updates the statistics of the execution times and communication overhead of the local and remote task executions to be updated in the database for ensuring accuracy in the future task offloading events. Listing 5.7 presents the *getExecutions* helper method in the database that returns past local and remote executions. All these previous executions were added to the database in the post-execution phase, as explained in Subsection 5.5.4.

The decision engine fetches a list of both local and remote executions to be used in the decision making algorithm, as shown in Listing 5.8.

5.7 MAMoC Server

MAMoC Server runs on ENs and PNs. It is implemented using Python with Java code execution support using a JVM interpreter. The three modules of the server runtime environment include a router, a server manager, and a repository.

In MAMoC-enabled mobile applications, Each task is identified by a unique ID which will be looked up in the remote server to check if it is previously been offloaded. All the annotated (offloadable) tasks are indexed and saved in a metafile during the launch of the application. This allows for easy retrieval of the source code of the task when it is needed to be sent over to the remote server. This procedure depends on the offloading service provider. If the offloading execution location is a nearby mobile device, we will simply use Java Reflect to execute the task in the connected mobile device. However, if the location selected by the offloading decision engine is an edge device or a public cloud instance, we need to retrieve and send over the Java source code of the offloaded task.

5.7.1 MAMoC Router

As the system architectures and the Internet of Things continue to push us towards distributed logic, we need a mechanism to route the traffic between those various components. The router works as a communication manager which is deployed on the MAMoC Router that runs on a separate container than the server manager as depicted in Figure 5.1.

There are multiple client library and router implementations[22] for integrating WAMP into the client-server applications. MAMoC router uses a custom version of Crossbar [23] for routing the offloading requests from Host Mobile Devices. This choice is made due to the fact that this router implementation is done by the developers of the protocol. It has a well-supported community and clear documentation for customising the router component. It is the original implementation of the WAMP which combines RPC with Pub/Sub communication patterns into a single communication layer. It supports event-based RPC in a high-throughput and low-latency system. Autobahn [24] is the name of the client library implementations that are written for multiple languages including Java, Python, C++, and Javascript. AutobahnPython[25] is used in the server manager module for handling offloading execution events.

[22]https://wamp-proto.org/implementations.html
[23]https://crossbar.io
[24]https://crossbar.io/autobahn/
[25]https://github.com/crossbario/autobahn-python

Routers are the core facilities of Crossbar, responsible for routing WAMP RPC between callers and callees, and routing Pub/Sub events between publishers and subscribers. This allows a computing node to seamlessly interact with the local infrastructure available. This mechanism is implemented by establishing a control channel for command streams and monitoring services based on WebSocket. WAMP also supports asynchronous transport and delivery system for message-encapsulated commands inheriting from WebSocket as a sub-protocol. Thus, it provides a full-duplex TCP communication channel over a single HTTP-based persistent connection.

5.7.2 Server Manager

This module holds the core components of MAMoC Server for handling the task remote executions and application refactoring processes.

5.7.2.1 Execution Controller

The main objective of the MAMoC Server is to be able to successfully execute the received tasks from mobile devices. After the request is validated in MAMoC router, the request is passed to the execution controller in the server manager module. As mentioned before, the server manager is bundled with a JVM to compile and run the Java codes. The server caches the source codes and resources files that are received from the mobile devices. Data caching is proven to improve data efficiency and response times in MCC solutions [67]. Thus, the execution controller first checks if the source code is already available in MAMoC repository module. Otherwise, the received source code is passed to the code transformer component described in Section 5.7.2.2.

This component is developed on top of AutobahnPython. Autobahn provides support for two asynchronous Python libraries, including Twisted[26] and Asyncio[27].

5.7.2.2 Code Transformer

As described earlier in Subsection 5.6.2, the code decompiler component in MAMoC Client is responsible for generating the Java source code from the Dalvik Executable format (DEX) files in the application. Even though the received source code is in Java format, it cannot be compiled with pure JVM yet. Therefore, this component statically generates a JVM-compatible Java code for the execution controller.

[26]https://twistedmatrix.com
[27]https://docs.python.org/3/library/asyncio.html

Algorithm 5.1 shows the steps taken on the decompiled source code that is received by the client. Some of the steps involve removing unnecessary code and adding Java pure codes that are not present on Android. In case the task is resource-dependent, a Java resource file reading code needs to be added as well.

Algorithm 5.1 Decompiled Android code transformation algorithm in the server

Input: sourceCode, resourceName, parameters
Output: result, duration

```
1: start ← startTimer()
2: code ← removePackageName(sourceCode)
3: code ← removeAnnotations(code)
4: className ← findClassName(code)
5: code ← addMainMethod(code)
6: if resourceName is not empty then
7:     code ← addResourceCode(code)
8: end if
9:
10: result ← executeCode(className, code, parameters)
11: duration ← endTimer() - start
12: return result, duration
```

Examples of code transformation are provided in Appendix A

5.7.2.3 Status Broadcaster

It was shown previously in Eq. (4.15) and Eq. (4.16) that the computation power of the ENs and PNs is an essential part of the task offloading decision making. A lightweight tool called *psutil (process and system utilities)*[28] is used for profiling the remote servers for computation power, available memory and battery information and publish it to the subscribed mobile devices.

```
import psutil
class StatsCollector(object):
    @classmethod
    def fetchstats(cls):
        cpu = psutil.cpu_freq().max * psutil.cpu_count()
        mem = round(psutil.virtual_memory().total / 1000000)
        battery = psutil.sensors_battery().percent

        return cpu, mem, battery
```

[28]https://pypi.org/project/psutil/

```
cpu, mem, battery = StatsCollector.fetchstats()
self.publish('uk.ac.standrews.cs.mamoc.stats', cpu, mem, battery)
```

Listing 5.9 Fetching and broadcasting server status

5.7.2.4 Application Refactor

An Android application is packaged as an APK (Android Package) file, which is essentially a ZIP file containing the compiled code, the resources, signature, manifest and every other file the software needs in order to run on the mobile devices. One can simply run *unzip* command and extract the files in the APK file. It contains three files, including a manifest file *AndroidManifest.xml*, a resources index file *resources.arsc*, and *classes.dex* file that contains the Dalvik bytecode of the app as described in Section 5.6.2.2. The APK also includes two *assets* and RES folders that contain documents, media files, layout XML files, and custom fonts and styles.

To fetch detailed information about an app without running it, MAMoC uses an open source tool called AndroGuard [36] to disassemble and decompile Android apps. Similar to the DexDecompiler component in MAMoC Client, it uses Jadx to obtain the Java source code from the Dalvik bytecode classes. It is also capable of decompiling the AndroidManifest file to its original XML format.

```
apk_list = [...]

class ApplicationRefactor:

    @classmethod
    def refactor(cls, app_id):
        tic = time.time()
        download_apk(app_id)
        a, d, dx = AnalyzeAPK('APK_files/{}.apk'.format(app_id))
        offloadbables = IdentifyAPK.identify(a, dx)
        IdentifyAPK.AnnotateOffloadables(a, offloadbables)
        sign_apk(a)
        time_spent = time.time() - tic

    def main():
        for apk in apk_list:
            ApplicationRefactor.refactor(apk)
```

Listing 5.10 Application refactoring class in MAMoC server

We first analyse the bytecode of the application for discovering the parts worth offloading. The Listing 5.10 is used to iterate the classes and methods. We will then rewrite the bytecode to implement a special program structure supporting on-demand offloading, and finally generate two artefacts to be deployed onto an Android phone and the server, respectively. As the result, two artefacts are generated: a signed APK file which can be used for future mobile devices to install the refactored app and the offloadable class files saved in the server and registered as remote procedures so that the client does not need to publish it before calling the procedure. Refactoring is transparent to app developers and supports legacy apps without source code.

There are no official API endpoints for fetching the list of Android native classes. It is possible to scrape the official Android package web pages to get a list of the Android native packages. Three three official package pages are: Android platform packages [29], Android support packages [30], Android wearable packages [31]. The retrieved package names are then saved in three files (android_platform_packages.txt, android_wearable_packages.txt, android_support_packages.txt) to be read by the classifier.

Decompiling to Java is lossy, meaning the code is probably good enough to get the gist of what it is doing but not good enough to re-build the APK. An APK decompiled to Smali with APKTool can be modified (change resources, inject code, modify code) and then built into a working APK again.

In summary, the refactoring bytecode steps are the following:

1. **Download the APK file:** the server needs the app ID to download a copy of the APK file of the mobile application.

2. **Decompile the APK file:** Using a third-party tool, the APK file can be decompiled to source code and resource files.

3. **Identify offloadables:** Checking which classes or methods are offloadable and which ones are unoffloadable. The unoffloadable classes use special resources available only on the phone, such as the GUI (Graphic User Interface) displaying, the camera, the acceleration sensor, and other sensors. If being offloaded to the server, these native classes cannot work because the required resources become unavailable.

[29]https://developer.android.com/reference/packages.html
[30]https://developer.android.com/reference/android/support/packages.html
[31]https://developer.android.com/reference/android/support/wearable/packages.html

4. **Annotate the offloadable classes and methods:** when the class or the method is identified as offloadable, an *@Offloadable* annotation needs to be added to it.

5. **Group and cache the offloadable classes:** The Java source code and bytecode files of the app and the reference resources files, e.g., images, data files, and jar libs are cached in the MAMoC repository for future task offloading requests.

6. **Generate the new APK file:** The refactored Android app will be compressed into an .apk file. It will then be available to mobile phones to install. A publish message will be sent back to the mobile phone with the link of installation of the APK file.

5.7.3 MAMoC Repository

This module acts as a storage for the code and resource files that are processed in the server manager or received from the mobile devices.

The repository caches the following types of files:

- APK Files

  The downloaded APK files are cached in the repository in case a new application refactor is needed in the future.

- Java Files

  The Java source code files and bytcode classes are cached in the repository after the task is registered in the server manager. The future RPC calls to this task will result in fetching the cached files with the newly passed parameters.

- Resource Files

  The tasks that are resource-dependent need data to be present in the offloading site. The task includes a hash of the resource file. The cached file is used for executing the task if the hash is the same as the hash of the cached file. Otherwise, the server manager asks for new resource files to be fetched from the mobile device. MAMoC assumes that the resource files are already present in the offloading sites, so this work is currently not implemented in the system and left as a future work.

5.8 Integrating MAMoC Client to Existing Projects

The simplicity and ease of integration of the framework has been the priority in designing and implementing MAMoC. The client offloading library is made public to allow mobile application developers to include it in their Gradle file and make their apps MAMoC ready to start discovering services and offloading tasks.

A simple example of counting prime numbers is used to demonstrate the steps of integrating MAMoC into an existing Android application. The Android class *PrimeCounter.java*, which is written in Java programming language, counts the number of prime numbers between one and a given number n as shown in Listing 5.11.

```
public class PrimeCounter {
    long n;

    public PrimeCounter(long n){
        this.n = n;
    }

    public long run() {
        int count = 0;
        for(int number = 2; number < n; number++){
            if(isPrime(number)){
                count++;
            }
        }
        return count;
    }

    private boolean isPrime(int number){
        for(int i=2; i < (number/2); i++){
            if(number % i == 0){
                return false; // number is divisible so it is not
                    prime
            }
        }
        return true; // number is prime now
    }
}
```

Listing 5.11 Prime counter Java class example

In order to make the above class compatible with MAMoC offloading library, the developer needs to follow the steps below:

1. Include the MAMoC client[32] in the *build.gradle* file of the Android app module:

   dependencies implementation 'uk.ac.standrews.cs:mamoc_client:0.14'

2. Add the *@offloadable* annotation to the class. If the class needs input files, set *resourceDependent* option to *true*. If the task can be independently parallelized into subtasks, set *parallelizable* option to *true*.

```
import uk.ac.standrews.cs.mamoc_client.Annotation.Offloadable;

@Offloadable(resourceDependent = false, parallelizable = false)
public class PrimeCounter {
...
```

Listing 5.12 @Offloadable interface to annotate the compute-intensive tasks

3. In order to allow calls for the offloadable class, an *Activity* can be created to contain the calling method of the class.

4. The framework needs to be initialized before calling the offloadable tasks. Listing 5.13 create a singleton of the MamocFramework class which works as a central manager for all the other components in the framework. This code can be added to the *Application.java* class or in the *onCreate* method of the Activity to get an instance of the framework and initialize its components.

```
mamocFramework = MamocFramework.getInstance(this);
mamocFramework.start();
```

Listing 5.13 Initializing the MAMoC framework

5. Since MAMoC broadcasts the results of the execution to the application, the activity or the calling class of the offloadable task has to register the local broadcast object to receive the results, duration, and communication overheads of the task execution as shown in Listing 5.14.

```
String OFFLOADING_RESULT_SUB = "uk.ac.standrews.cs.mamoc.
    offloadingresult." + PrimeCounter.class.getName();
LocalBroadcastManager.getInstance(this).registerReceiver(
    mMessageReceiver, new IntentFilter(OFFLOADING_RESULT_SUB));

    protected BroadcastReceiver mMessageReceiver = new
        BroadcastReceiver() {
        @Override
```

[32] https://bintray.com/dawand/mamoc_client

```
        public void onReceive(Context context, Intent intent) {
            String result = intent.getStringExtra("result");
            Double duration = intent.getDoubleExtra("duration",
                0.0);
            Double overhead = intent.getDoubleExtra("overhead",
                0.0);
            ...
        }
    };
```

Listing 5.14 Local broadcast registration for receiving the offloaded task execution result

6. Finally, the task name and any necessary parameters can be passed to the framework with the option to specify the location of the execution (*LOCAL*: Locally, *NEARBY*: on a nearby mobile device, *EDGE*: on an edge node, *PUBLIC_CLOUD*: on a public cloud instance, or *DYNAMIC*: let MAMoC decide the execution location). The default option is DYNAMIC.

```
String taskName = PrimeCounter.class.getName();
long n = 100000;
mamocFramework.execute(ExecutionLocation.DYNAMIC, taskName, n);
```

Listing 5.15 Executing the task by specifying the execution location

In order to create a user facing activity with all the offloading scenario options, an example of *SearchActivity.java* on *MAMoC-Demo* repository [/src/main/-java/uk/ac/standrews/cs/mamoc_demo/SearchText/SearchActivity.java] can be adapted. If there are more activities in the same application, a base activity can be used to work as a parent class for all the activities similar to *DemoBaseActivity.java* [src/main/java/uk/ac/standrews/cs/mamoc_demo/DemoBaseActivity.java] which includes the common methods used by the activities in performing the offloading operations and listening to local and remote executions performed by MAMoC.

5.9 Summary

This chapter provided an overview of MAMoC system and its architectural design. The proposed MCO system comprises MAMoC Client and MAMoC Server with each of them involving many loosely coupled services that interact with each other in order to perform supported features. Section 5.2 specified the assumptions in terms of communication and execution prospects in order for MAMoC to behave as expected. A concise summary of each module and its corresponding components are presented in Section 5.3. The secured interaction between components is achieved using the communication mechanisms depicted in Section 5.4. To understand the execution life cycle of the offloadable tasks on MAMoC-enabled mobile applications, Section 5.5 presented the four phases starting from the preparation phase until the return of the result of the task execution.

The overall implementation corresponds to the architectural design discussed in the first part of this chapter for both the client and server components. The implementation is intended to meet the requirements stated in the previous chapter. MAMoC is unique in its loosely decoupling of the components in both the client and server-side modules. Furthermore, MAMoC exposes a novel programming model which allows data-rich and compute-intensive tasks in mobile devices to be transparently offloaded. These strengths, amongst others, make MAMoC an ideal candidate for mobile cloud offloading and a suitable replacement. The subsequent chapter tests different components and analyses the execution results.

Chapter 6

Experimental Evaluation

6.1 Overview

The main goals of the proposed MAMoC framework are to allow mobile application developers to achieve a transparent automated offloading to multiple destination clouds and device dynamic changes over the life cycle of execution of an application. MAMoC has been designed to improve the execution time of mobile apps through better offload decision making. This results in reduced energy consumption of the local device and improved responsiveness of its applications.

This chapter describes a series of experiments that were conducted to exercise the proposed methodology and validate the measurements. Four different sets of experiments are conducted to thoroughly evaluate the different components of the framework. Subsection 6.1.1 describes the process of provisioning and deploying the participating nodes in the evaluation testbeds.

Section 6.2 includes the first experiment to evaluate the performance of the offloading decision algorithm used in MAMoC decision engine. The completion time and energy consumption of the tasks are measured locally or remotely with different offloading sites and an external offloading system. Then, Section 6.3 describes the task partitioning mechanism based on the site offloading scores used for parallizable tasks that are both data and compute-intensive with different offloading scenarios. Section 6.4 demonstrates the MCDM approach described in Section 4.5 earlier that is used when multiple criteria rather than just execution speed are considered including bandwidth, availability, security and price of the offloading sites. Both single decision making and group decision making approaches are evaluated with extra offloading sites. Finally, Section 6.5 evaluates the application refactoring component in MAMoC Server for parsing and analysing APKs to identify the offloadable tasks and transparently annotate them.

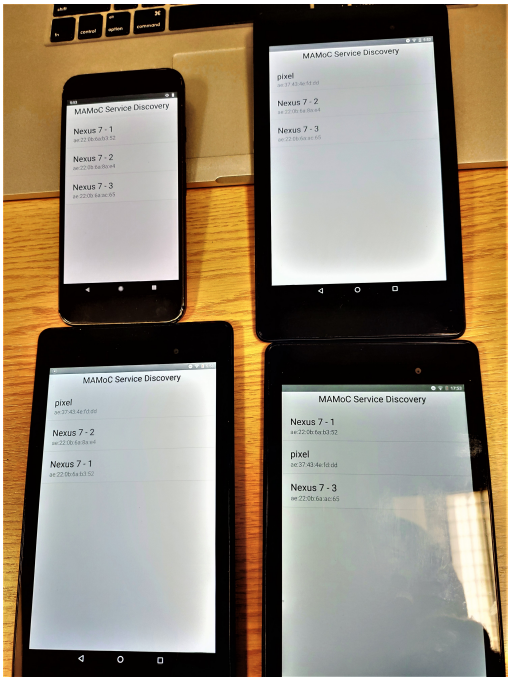


Fig. 6.1 Mobile Ad-hoc Cloud devices connected through WiFi-Direct

Each evaluation section contains results and analysis subsections to discuss the results attained from running the tests. The experiments are used to empirically evaluate the offloading model and methodology. In some cases, precise analytical models are presented and explained, accompanied by numerical results showing concrete figures of the achievable gains. Section 6.7 will summarise the overall results and list some limitations with the framework and the evaluation outcomes.

6.1.1 Setup and Deployment

MAMoC-Demo[1] Android mobile application is developed to contain the offloadable tasks which are employed for the experiments. This application is supported by the MAMoC Client offloading library, which includes the necessary client-side components of the framework. The application is installed on both mobile devices for working as both offloading host and service provider in the form of D2D offloading.

For the edge server used as a Cloudlet, a VM is installed on the KVM hypervisor on our school server with the following command:

[1]https://github.com/mamoc-repos/MAMoC-Demo

```
virt-install --connect qemu:///session --os-variant=ubuntu17.04 --network
default --network bridge=br0 --name EdgeServer --ram=16384 --vcpus=8
--cpu host-passthrough --disk pool=default,size=32 --rng /dev/random
--location "http://archive.ubuntu.com/ubuntu/dists/bionic/main/installer-
amd64/"
```

Since the servers cannot be accessed off-campus, a reverse proxy technique is used to provide access to the server even if the mobile device is not currently connected to CS network. The Websocket connection can be verified through the following curl command:

```
curl --include --no-buffer --header "Connection: Upgrade" --header "Up-
grade: websocket" https://djs21.host.cs.st-andrews.ac.uk/offload/ws/
```

The public cloud node is an AWS c4.4xlarge instance type. We chose the AWS region (London) with an average latency of 27ms from our school network to deploy the instance[2]. The instance is launched with a pre-booting script (called user data on AWS console) shown below:

```
#!/bin/bash
sudo yum update -y
sudo amazon-linux-extras install docker -y
sudo systemctl start docker
sudo usermod -aG docker ec2-user
docker pull dawan/mamoc_router
docker run -itd --name "mamoc-router" -p 8080:8080 dawan/mamoc_router
docker pull dawan/mamoc_server
docker run -itd --name "mamoc-server" --network="host" dawan/-
mamoc_server
```

The energy and power consumption data are derived from direct measurement of the Android device (Nexus 7). More specifically, Qualcomm's Android app, Trepn profiler was used for profiling. All the benchmark experimental results resented here are with Deltas enabled. When profiling with Deltas enabled the app profiles (collects power data) for the entire system for a baselining interval and

[2]https://www.cloudping.info/

then subtracts the average value of power, so obtained, from all subsequent raw values. All the experiments were conducted with a maximum possible baselining period of 30 seconds with a wake lock for the entire period of profiling. The power profiles of the application were saved as CSV files, which were then processed offline to compute the energy consumed. Studies report that software tools such as the Trepn Profiler [98] used here indicate they can be as accurate as methods based on external hardware measurement devices such as the Monsoon power meter [94] but at a much lower cost[3].

6.2 Offloading Decision Algorithm Evaluation

This section evaluates the offloading decision algorithm listed in Algorithm 4.1 and explained in detail in Chapter 4. The experiments have been carried out through a real-world testbed deployment described in Subsection 6.2.1. The MAMoC-enabled demo application, which includes all the offloadable tasks, is described in Subsection 6.2.2. Then, Subsection 6.2.3 discusses the results of executing the tasks in terms of the completion time of the tasks and energy consumption of the host mobile device. An external offloading library called ULOOF [96] is also used to compare the results of running the demo applications as shown in Subsection 6.2.4.

This experiment was carried out as part of our published paper [137] in which the same offloadable tasks in the demo application and experimental methodologies were used.

6.2.1 Experimental Environment

This experiment considers a scenario where a mobile app running on the host mobile device (Self Node) is associated with a nearby mobile device as Mobile Node, an edge server as Edge Node deployed in LAN, and a public cloud instance as Public Node to improve energy and performance on the users' devices. This test is performed on two stock Android mobile devices. For a slightly older and a lower-end device, a Nexus 7 tablet released in 2013 with a Quad-core 1.5 GHz CPU and 2GB of RAM as a Self Node; For a high-end mobile device, a Google Pixel phone with a Quad-core 2.15 GHz CPU and 4GB of RAM is used as a Mobile Node. The detailed specifications of the testbed devices are shown in Table 6.1.

[3]https://mostly-tech.com/2015/05/28/

Node	CPU (in GHz)	RAM (in GB)	OS	Location
Nexus 7	1.3 (Dual)	1	Android 6.0 (Marshamallow)	St Andrews, UK
Pixel	2.15 (Quad)	2	Android 9.0 (Pie)	St Andrews, UK
Macbook Pro	2.5 (Quad)	16	Mac OS 10.13	St Andrews, UK
c5.4xlarge	3.0 (16-core)	64	Ubuntu Server 18.04	London, UK

Table 6.1 Device specifications for the offloading decision algorithm evaluation

6.2.2 Demo Application

The aforementioned application includes the following tasks:

- **Text Search:** It allows a user to enter a keyword and select a file size from (small, medium, and large) to find the occurrences of the word in the file. Knuth-Morris-Pratt string searching algorithm [72] is used. This is an example of an embarrassingly parallel task, since it can be independently run on multiple nodes; Hence, the *parallelizable* annotation optional value is set to true. This was demonstrated in the previous experiment. The external node that performs a full or partial search should have access to the text file so the file needs to be transferred over hence the *resourceDependent* is also set to true. The entered keyword in the mobile device by the user also needs to be sent over as a parameter to the remote resource.

- **Quick sort:** Quicksort algorithm is used as the sorting algorithm to sort the words of a text file. The mobile device or the service provider first needs to fetch the content of the text file and apply the sorting algorithm to the list of space-separated words. It does not require any extra parameters to be sent from the host mobile device.

- **N-Queens:** The task is to enumerate the placements of all N valid queens on an N x N chessboard such that no queen is in the range of another. 6 different N value tests from $N = 8$ to $N = 13$ are used for the lower and higher computation intensity of the task. It does not depend on any external resource, so only the N (number of queens) parameter needs to be sent over in all the offloading scenarios.

Text search and sorting examples are data-dependent. Three common files with different sizes are used. The large text file consists of 1,095,649 words, the

medium text file contains 316,323 words, and the small text file contains 39,799 words. However, the files are not partitioned in this evaluation since all the executions are done in a non-parallel fashion. The files are stored in the mobile application running on the Host Mobile Device and they are published to the offloading sites with the first offloading request and cached in MAMoC repository in the MAMoC Server for future requests assuming there will be no changes in the content of the files.

These selected tasks are commonly used in real-world applications in the literature [130] which surveys the benchmarking applications used in MCC research works. GNU Grep [4] is a program that searches through a file for lines which contains a given keyword. The I/O-bound module in Grep finds desired lines in a file; the CPU-bound module in Grep transfers keywords and file names to the I/O-bound module. Word Count in GNU Coreutils [5] counts the number of words in a set of files. In the next evaluation set, our framework partitions the text file similar to a word count application into an I/O-bound module that calculates word occurrences in one file, and a CPU-bound module that sums the occurrences up.

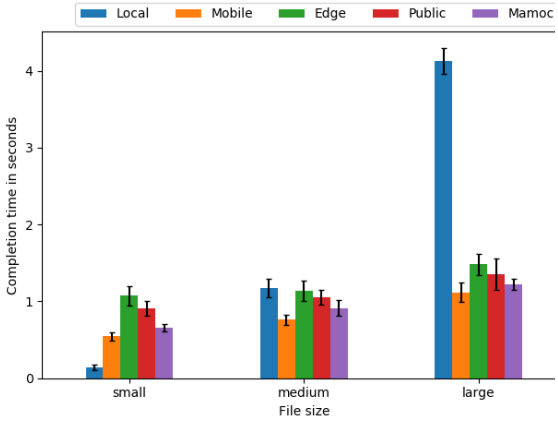
GNU core utilities also contains a sort application along with many other programs. The sort application uses merge sort which sorts the lines of a text file in alphabetical order. Similar to this, our sorting task treats the entire sort application as an offloaded I/O bound module that receives a file name and stores sorted text in a file.

More real-world applications are discussed in [130] that categorises them into imaging tools such as face identification and recognition and OCR, mathematical tools such as Linkpack, sorting algorithms, and Fibonacci, games such as chess, and word processing applications . Similar to our approach, word count and sort examples are used in [156] with different input sizes to evaluate their completion time and energy consumption in local and remote execution scenarios. N-Queens is a common CPU benchmarking tool used in multiple research works in the literature including [73], [74], and [46].

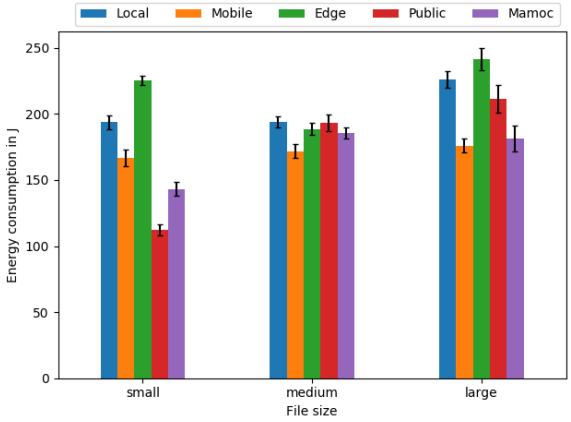
6.2.3 Results and Analysis

The three example tasks in the demo application are executed 30 times each and plotted an average for both the completion time in seconds and energy consumption in Joules. The Maximum Local Executions ($MaxLE$) and Maximum

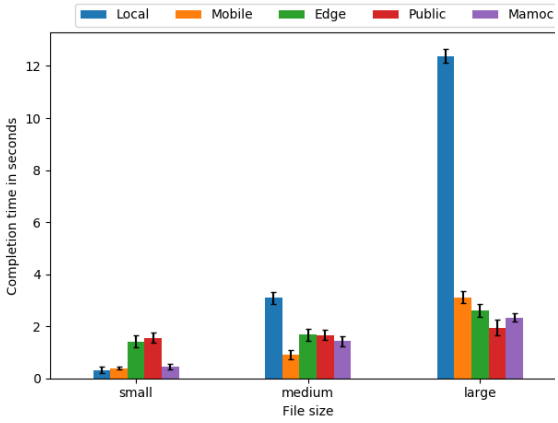
[4]http://www.gnu.org/software/grep/
[5]http://www.gnu.org/software/coreutils/



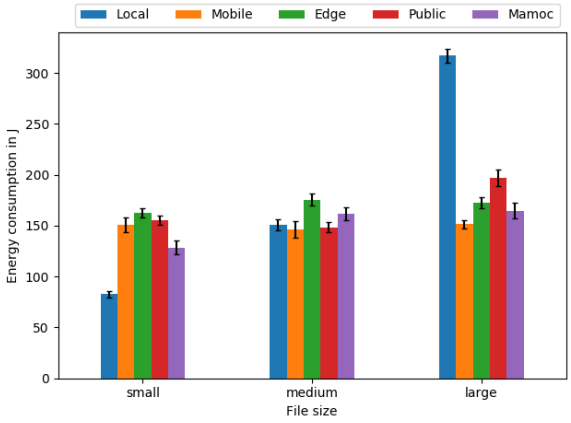
(a) Text search completion time



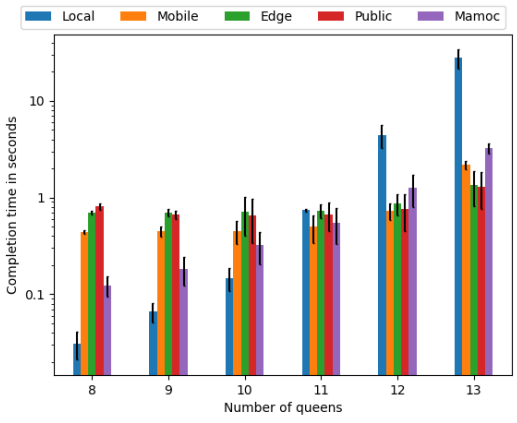
(b) Text search energy consumption



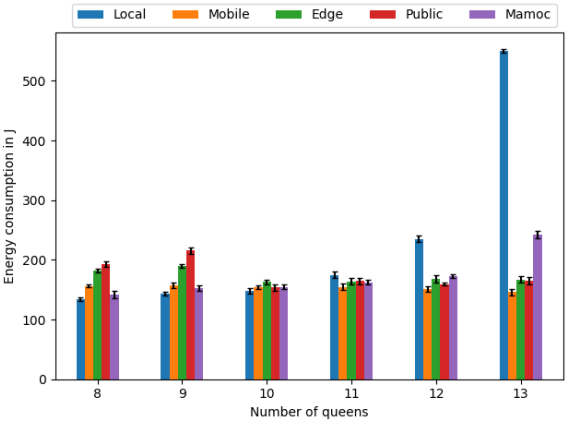
(c) Quick sort completion time



(d) Quick sort energy consumption



(e) N-Queens completion time



(f) N-Queens energy consumption

Fig. 6.2 The demo application task execution results in the offloading decision algorithm evaluation

Remote Executions ($MaxRE$) execution checkpoint variables are key points in deciding to perform a remote execution or fall back to local execution as shown in Algorithm 4.1. The gains in response time and energy obtained by using MAMoC are shown in Figure 6.2. The proposed multi-criteria solver algorithm then selects the offloading site after performing the AHP and fuzzy TOPSIS methods listed in Algorithm 4.4. A detailed MCDM evaluation is left for the next set of evaluation where both the single decision making that is used for this evaluation and group decision making tests will be conducted.

6.2.3.1 Completion Time

Completion time of the offloading task to a remote node contains communication time in the wireless network and computation time in the offloading site. As the task data size (DT_{T_i}) increases, the communication time and computation time are increasing. From Figure 6.2a, it can be observed that in terms of total completion time, the local execution is preferable for a small text file but not in the case of medium or large text files. The minimum completion time for medium and large text file scenarios was when the task was offloaded to the nearby mobile device due to low transmission overhead and higher computation capabilities. As shown earlier in Algorithm 4.1, MAMoC checks if the task has been executed previously from the database entries of that task and its configurations. In this step, a simple heuristic is applied to check if the task has been executed for 5 times in a row in the same location ($MaxLE\%5 == 0$), e.g. edge, then MAMoC decides to run it on the other site, e.g. locally. By doing this, there is a mechanism to compare the executions and figure out if the task performs better locally for small input values and if for larger input values it is more convenient to offload its execution to the remote site. Similarly, in the Quicksort example in Figure 6.2c, it can be noted that the edge server and public cloud instance have shorter completion times. A similar pattern can be concluded in the N-Queens example in Figure 6.2e.

6.2.3.2 Energy Consumption

This represents the energy consumed by the host mobile device when the task is executed locally or remotely. Power estimation is dependent on the frequency and the load of each CPU core, the Graphics Processing Unit, and the brightness of the screen. Energy consumption for both computation and communication [16] are recorded during the execution of the task.

The two methodologies for measuring the energy consumption of mobile devices are hardware and software solutions [55]. Monsoon [94] is a well-known

power monitor used in many mobile computation offloading systems. There are a good deal of software-based mobile device power modelling and analysis tools [52][160][100] that are used in the mobile computation offloading literature. Trepn Profiler [98] is an on-device standalone profiling tool that displays an overlay UI with real-time graphs for CPU loads and battery data. App-specific power consumption and utilization can be saved in a CSV file in the mobile device. The file can then be exported to a desktop computer for offline analysis. The energy consumption figures are generated using this approach. One of the concerns of this profiler is that it only works on Snapdragon chipset-based Android devices powered with special component-wise sense resistors and power management IC. In a survey on software energy profilers [57], it is shown that Trepn profiler can achieve up to 99% accuracy against the power measurement results with the external devices.

Figure 6.2b, Figure 6.2d, and Figure 6.2f depict the energy consumption variance with the task size. It is observed that the proposed scheme outperforms local computing and full offloading to the public cloud server. In the meantime, the offloading method gets a better result, especially when the task size becomes larger. Therefore, for large computing tasks, the algorithm prefers to offload large partial computation tasks to the more powerful sites to reduce mobile consumption. The full offloading method is expected to have better performance than local computing on energy consumption. Since the proposed approach considers the trade-off between the advantages of local computing and full offloading methods, it reduces energy consumption in total.

It can also be noted that in some scenario executions, MAMoC has a longer average completion time and energy consumption than a particular offloading site. Nonetheless, in the long run, it can adapt to the dynamic environment changes and make better decision making than always selecting local execution or full offloading to a single site.

6.2.4 Comparative Evaluation

In this section, the performance of MAMoC will be compared to an external mobile computation offloading framework called *ULOOF* (User-Level Online Offloading Framework) [96].

Similar to MAMoC, *ULOOF* uses an offloading decision engine to decide whether a method should be offloaded or executed locally without modifying the mobile device's operating system. The networking and device profilers are also monitoring changes in the Host Mobile Device battery status and network

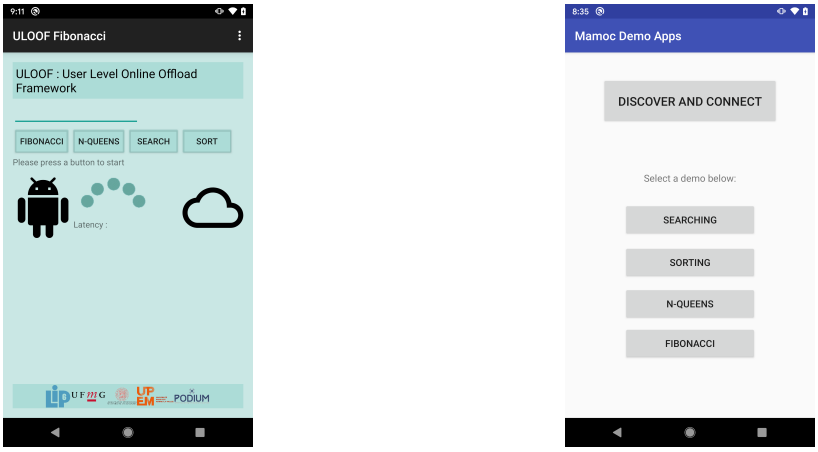
connection. *@OffloadCandidate* annotation is used to identify the offloadable tasks. The decision engine always offloads the first two executions of a task if the offloading is enabled and the device is connected to an offloading server. It will then fall back to local execution for the second two executions. The decision to offload will then be taken according to the mean values of the local and remote execution times. To accommodate message communications between the mobile device and offloading server, it uses Kryo[6] which is a popular binary serialization library to transfer the offloaded method and their arguments.

The source code for the offloading library and offloading server (which is also run on Nearby Mobile Devices) is available on Github[7]. The source code had some issues and many segments of the code were commented out. I modified some of the issues in the framework without changing the decision engine logic. In order to setup the comparative evaluation testbed, I performed the following necessary changes to *ULOOF*:

- The framework only worked when a server was already connected. It now falls back to local execution when the connection to offloading server is not available.
- The network profiler used to measure round trip times even when there is no connected server. This is changed to only calling getRTT() if the server is available.
- Adding three new tasks (explored in the previous subsection) along with the original Fibonacci code. This also include updating the user interface components to connect them to the list of input sizes for automating the execution of the different benchmarking tasks.

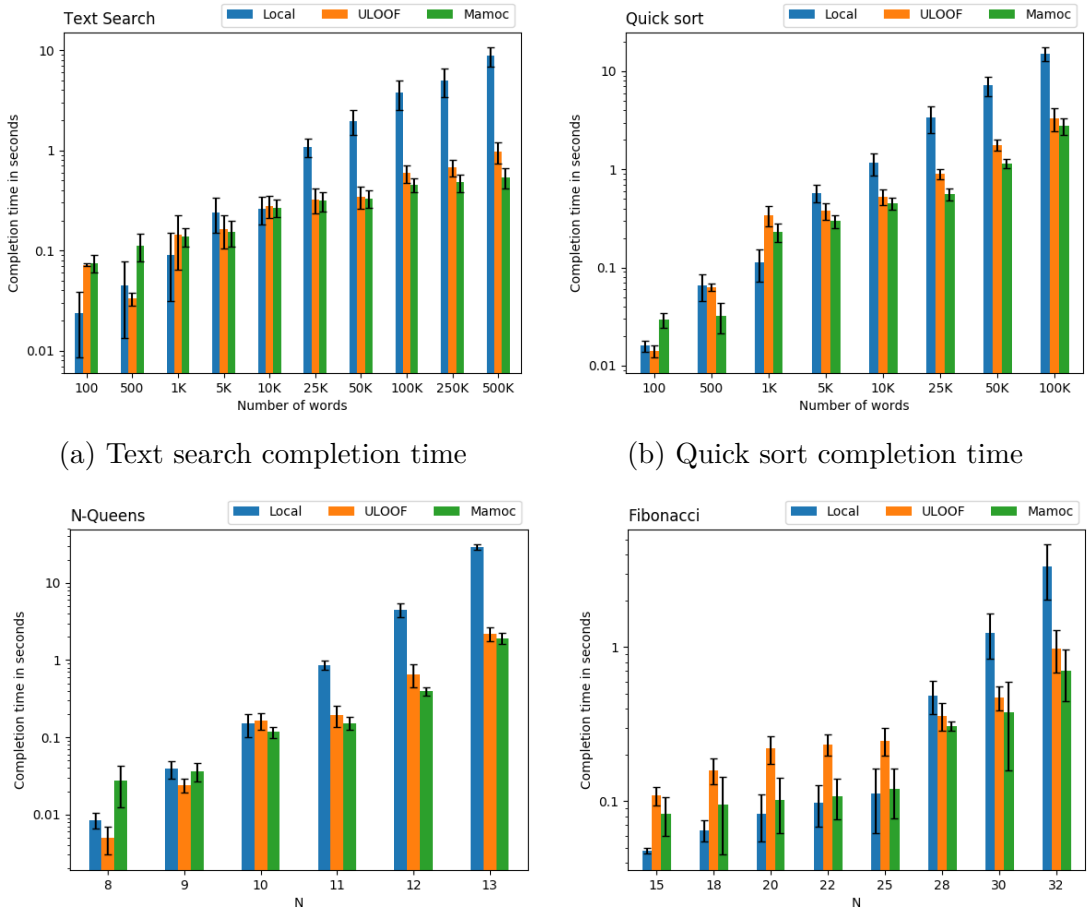
ULOOF already contains a benchmarking test of calculating Fibonacci. This method is tested with different arguments including sending the method a single integer value or a list of integer values. I have added the three demo applications to their testing application as shown in Figure 6.3a. The Fibonacci code is also added to our testing application to compare results of all the four benchmarking results. The main activity user interface is shown for the MAMoC demo application in Figure 6.3b.

[6]https://github.com/EsotericSoftware/kryo
[7]https://github.com/ULOOF-Framework/ULOOF



(a) ULOOF demo application main interface (b) MAMoC demo application main interface

Fig. 6.3 Demo applications of both MAMoC and ULOOF for conducting the comparative evaluation



(a) Text search completion time (b) Quick sort completion time

(c) N-Queens completion time (d) Fibonacci completion time

Fig. 6.4 Comparative evaluation results for the demo applications in local, MAMoC, and ULOOF executions

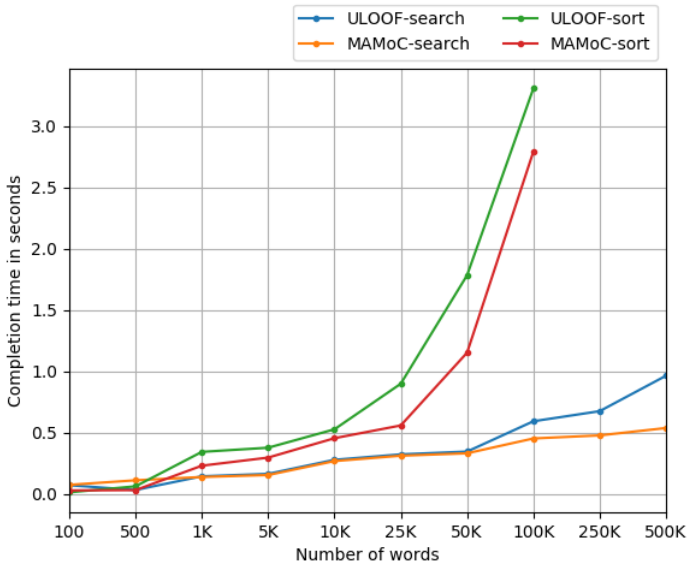


Fig. 6.5 Incremental comparison between ULOOF and MAMoC completion times for text search and sorting tasks

ULOOF's offloading decision engine always offloads the first two executions and executes the second two executions locally. It will then compare the mean times of local and remote execution times and make offloading decisions accordingly. The experiments are run 10 times and an average of the completion times is calculated for each scenario.

It can be noticed from the Figure 6.4a that the local execution only performs well under approximately 25,000 words where a decision engine can outperform it. This was also observed in the text search evaluation in the previous section. The same hypothesis holds true for other tasks shown in Figure 6.4. The difference in completion times between ULOOF and MAMoC starts to appear after the file size gets larger where in the largest file size MAMoC performs 15% better than ULOOF as shown in Figure 6.5.

6.3 Task Partitioning Evaluation

As described in Chapter 3, partitioning applications to local and remote parts is one of the most researched topics in multisite offloading studies. While some works study the partitioning of the whole application to local and remote parts, some perform the partitioning at the finer-grained level. In this section, the example task of text search is employed as a parallizable and resource-dependent task for evaluating it in both single-site and multisite scenarios.

6.3.1 Experimental Environment

The testbed for this experiment consists of two mobile devices (a Nexus 7 and a Google Pixel phone), one edge server, and three public cloud instance types. The hardware specifications are shown in Table 6.2. All the experiments are performed in the Jack Cole building in the School of Computer Science at the University of St Andrews. The mobile devices are connected to "CS" network. The edge server is running as a VM in a server with dual quad-core Xeon processors (Intel Xeon E5645 @2.40GHz). For the remote cloud instances, three different Amazon Web Services instance types are used: small (t2.micro), medium (c4.xLarge), and large (c4.2xlarge). The two phones will be running the mobile application built on top of MAMoC Client while the edge server and public cloud instances will be running the containers of the MAMoC Server modules.

Node	**CPU (in GHz)**	**RAM (in GB)**	**OS**	**Location**
Nexus 7	1.3 (Dual)	1	Android 6.0 (Marshamallow)	St Andrews, UK
Pixel	2.15 (Quad)	2	Android 9.0 (Pie)	St Andrews, UK
Cloudlet	2.4 (Quad)	16	Ubuntu Server 17.04 LTS	St Andrews, UK
Cloud (small)	2.4 (Single)	1	Amazon Linux 2 [8]	London, UK
Cloud (medium)	2.8 (Quad)	7.5	Amazon Linux 2	London, UK
Cloud (large)	2.8 (Octa)	15	Amazon Linux 2	London, UK

Table 6.2 Experimental environment device specifications for the offloading score evaluation

[8]https://aws.amazon.com/amazon-linux-2/

The mobile application which is used to test the performance of the framework and showcase the different execution scenarios contains a Knuth-Morris-Pratt searching algorithm [72] to be performed on three different size text files. This task is considered both data-intensive and compute-intensive for lower-end mobile devices. Three text file with different sizes and the number of words are used. The large text file consists of 4,382,596 words, the medium text file contains 1,266,492 words, and the small text file contains 318,392 words. The files are stored in the mobile application running on the Host Mobile Device and they are transferred to the offloading sites as a payload with each offloading request. This experiment measures the task completion time (M_{T_i}) on both the host mobile device in local execution and the offloading sites in remote execution modes. The application is executed in four different modes: local execution on the Self Node, on a nearby mobile device (Mobile Node), on a cloudlet server (Edge Node), and on public cloud instances with three different server configurations (Public Node). Four offloading scenarios are used: full offloading and two types of partial offloading (workload sharing in a parallel manner) with different configurations. Each execution was repeated 30 times, such that averages could be calculated for more accurate results.

6.3.2 Offloading Scenarios

MAMoC enhances the offloading process by considering the parallel execution of the application's independent tasks in the local device and offloading sites. In all the scenarios, the Nexus 7 device is the Self Node and the rest of the nodes are service providers (offloading sites) with the Pixel being the Mobile Node, the edge server acting as Edge Node and the public cloud instances as PNs. The offloading scenarios are described in the following subsections.

6.3.2.1 Full offloading

In full offloading mode, the execution is performed at the offloading site and the final result is returned to Host Mobile Device. Conversely, the partial offloading mode only sends a part of the execution over to the site and performs the rest of the execution itself as can be observed in the other two scenarios. After the results are received, they are merged and stored in the mobile device.

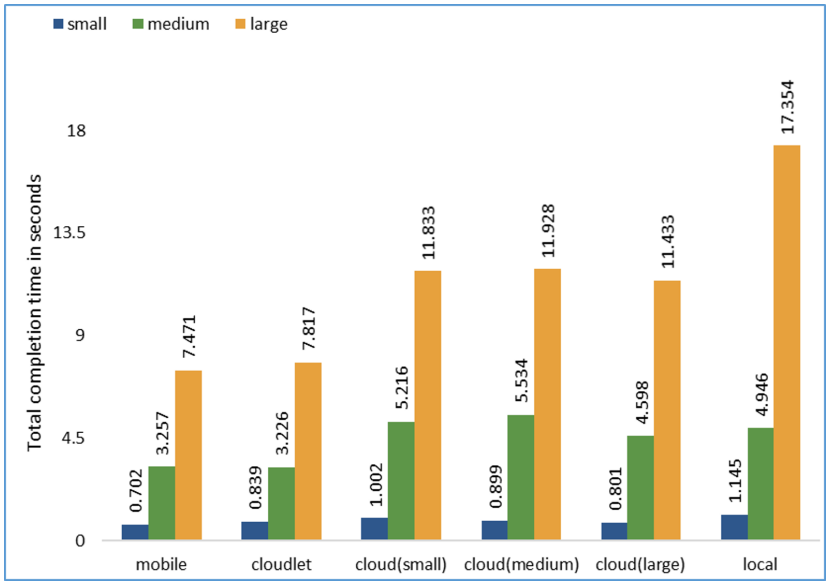


Fig. 6.6 Full offloading: the whole computation and payload are offloaded to the offloading site

MAMoC is specifically designed to support partial offloading in a parallel multisite fashion. Nonetheless, we wanted to observe the completion times of full offloading executions and compare them to the decision engine results. The full offloading scenario is performed on each offloading site separately using the three text files. Figure 6.6 shows the total completion time of running the application with different text file sizes. As it can be observed later, most of the completion time is the communication overhead that occurs during the transmission of the necessary data (the text file content) from the mobile device to the offloading sites.

6.3.2.2 Partial offloading

The first set of partial offloading experiments are performed with no help from the MAMoC decision engine. The tasks are equally distributed among the connected nodes. In other words, if there is only one available offloading site, the workload is divided into two equivalent halves and distributed to them for local and remote executions. Both devices then execute the workload in a parallel fashion. The local result and the result returned from the site are then merged and stored locally. The results of running the same set of workloads as the previous experiment are displayed in Figure 6.7.

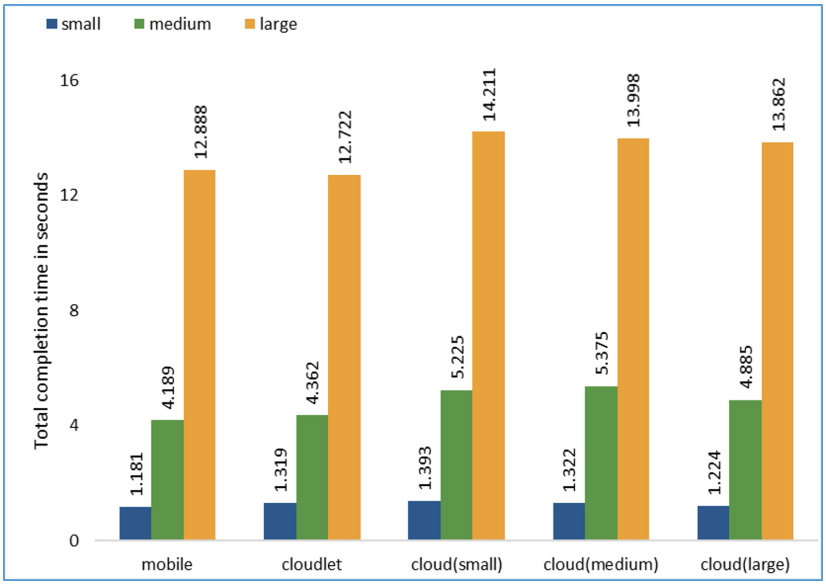


Fig. 6.7 Partial Offloading (equal task distribution) - Local mobile device executes 50% of the task while the remaining 50% is offloaded

6.3.2.3 Partial offloading with the decision engine support

The decision engine uses the offloading scores calculated in the models presented earlier to calculate the percentage of the task that should be offloaded to any particular offloading site. The offloading scores of the local device and the offloading sites are investigated by the decision engine for the task partitioning process. In the case of offloading score being less than zero, no computation is offloaded to that offloading site. The standardized and the process of calculating offloading scores are demonstrated in Table 6.3.

	Mobile	**Cloudlet**	**Cloud**	**Local**	**AVG**	**STD**
B	2.448	3.245	3.657	1.543	2.723	0.933
St. B	-0.295	0.559	1.001	-1.265		
CP	8.600	9.600	19.200	2.600	10.000	6.868
St. CP	-0.204	-0.058	1.339	-1.077		
RTT-small	0.605	0.703	0.949	-	0.752	0.177
St. RTT-small	-0.831	-0.278	1.110	-		
RTT-medium	2.860	3.040	5.490	-	2.848	2.246
St. RTT-medium	0.006	0.086	1.177	-		
RTT-large	4.730	5.460	11.240	-	5.358	4.608
St. RTT-large	-0.136	0.022	1.276	-		
OS-small	0.333	0.779	1.230	-2.342		
OS-medium	-0.504	0.415	1.163	-2.342		
OS-large	-0.363	0.479	1.064	-2.342		
OffPerc-small	14.19%	33.26%	52.53%	0%		
OffPerc-medium	0%	26.29%	73.71%	0%		
OffPerc-large	0%	31.03%	68.96%	0%		

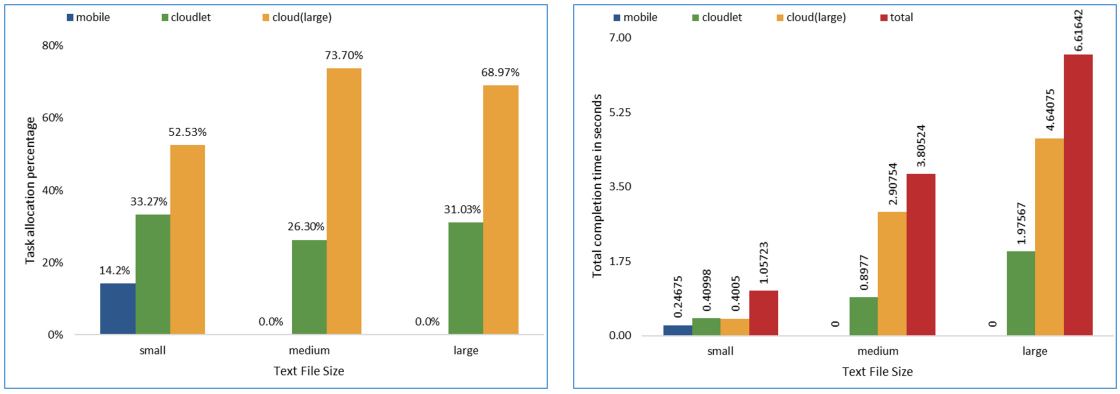
Table 6.3 Calculating offloading scores of the nodes for the task partitioning evaluation

The task partitioning percentage and completion times of partitioned tasks for multi-site offloading scenario are displayed in Figure 6.8a and Figure 6.8b accordingly. It is worth noting that only the large instance type of the public cloud node is used along with a cloudlet and a nearby mobile device as offloading sites in the multisite offloading scenario.

6.3.3 Results and Analysis

It is already shown in the literature that offloading does not always benefit the lower-end devices [77]. In the first execution scenario, the whole text file has to be transferred to the offloading destination to perform the search operation in the corresponding computation device. In the results shown in our work in [136] as shown in Figure 6.10, despite few millisecond performance gains in the case of the small text file, local execution was preferred to full offloading for medium and large text files. However, in this new testbed, full offloading is preferred in all the scenarios with offloading to the nearby mobile device being the best for small and large text files while the cloudlet performs the best for the medium text file.

In the results of [136], the partial offloading with equal partitioning of tasks among the local and external devices perform better in terms of reducing the overall network overhead occurrence in the previous execution scenario. However, because of the high local execution time of the host device in this testbed, the equal partitioning approach only increases the total completion time in all the scenarios. We also presented single-site partial offloading (where the task is meant to be executed locally and a single offloading site) that produced better results than equal task distribution scenario in all the offloading modes. In this evaluation, that was not performed since the offloading score of the host device was below zero so the results would be the same as full offloading to that particular offloading site.



(a) Multisite partial offloading - Task partition percentage

(b) Multisite partial offloading - Total completion time

Fig. 6.8 Multisite partial offloading evaluation results

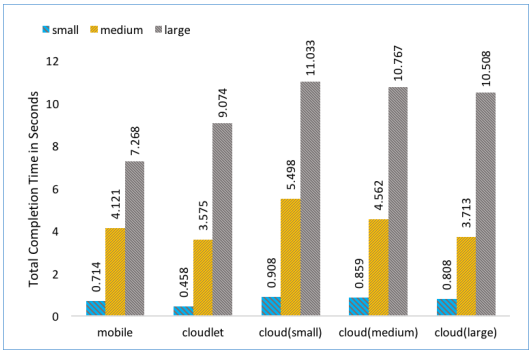
Finally, for the partial offloading with MAMoC decision engine support, it can be observed that no partitions are allocated to the host device due to its offloading score being below zero. For the small text file, the nearby mobile device, cloudlet, and the cloud nodes have distributively executed the task. The total completion time is not as good as some full offloading completion times but still performs better than the equal partitioning scenario. For the medium text file, the task is only partitioned to the cloudlet and cloud nodes, which similarly does not produce a lower completion time than the full offloading scenarios. However, for the large text file, the multisite approach performs better than all the full offloading scenarios including the minimum completion time on the nearby mobile device (6.616 vs 7.471).

File size	Local	Offloading site	Full offloading			Partial offloading (equal task partition percentage)				Partial offloading (multisite)			
	Comp.		Comp.	Comm.	Total	Local	Remote		Total	Task partition percentage	Remote		Total
							Comp.	Comm.			Comp.	Comm.	
Small	1.145	Mobile	0.097	0.605	**0.702**	0.885	0.072	0.224	**1.181**	14.2%	0.035	0.212	1.057
		Cloudlet	0.136	0.703	0.839	0.885	0.128	0.306	1.319	33.27%	0.058	0.352	
		Cloud (small)	0.115	0.887	1.002	0.885	0.112	0.396	1.393	-	-		
		Cloud (medium)	0.056	0.843	0.899	0.885	0.089	0.348	1.322	-	-		
		Cloud (large)	0.052	0.749	0.801	0.885	0.023	0.316	1.224	52.53%	0.026	0.375	
Medium	4.946	Mobile	0.397	2.86	3.257	2.943	0.225	1.021	**4.189**	0%	0	0	3.805
		Cloudlet	0.186	3.04	**3.226**	2.943	0.152	1.267	4.362	26.3%	0.047	0.851	
		Cloud (small)	0.196	5.02	5.216	2.943	0.099	2.183	5.225	-	-		
		Cloud (medium)	0.114	5.42	5.534	2.943	0.075	2.357	5.375	-	-		
		Cloud (large)	0.108	4.49	4.598	2.943	0.055	1.887	4.885	73.7%	0.079	2.829	
Large	17.354	Mobile	1.241	6.23	**7.471**	9.696	0.823	2.369	12.888	0%	0	0	**6.616**
		Cloudlet	0.357	7.46	7.817	9.696	0.232	2.794	**12.722**	31.03%	0.111	1.865	
		Cloud (small)	0.293	11.54	11.833	9.696	0.193	4.322	14.211	-	-		
		Cloud (medium)	0.198	11.73	11.928	9.696	0.142	4.160	13.998	-	-		
		Cloud (large)	0.193	11.24	11.433	9.696	0.123	4.043	13.862	68.97%	0.145	4.496	

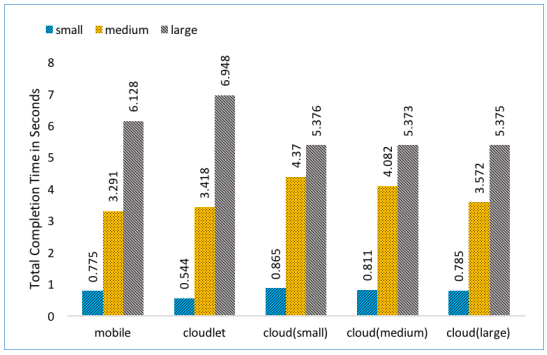
Fig. 6.9 The task partitioning evaluation results

For an overall reference of the results including the computation and communication breakdowns of the remote executions, a complete set of results of all the experiments are shown in Figure 6.9. It should be noted that this evaluation set is also demonstrated in our published paper [136] and presented in Figure 6.10. The difference between them is in the mobile platform choice. While the framework in the paper was implemented for iOS mobile devices[9]. Similar environments and the same offloading scenarios are performed again on the new MAMoC implementation for the Android platform. Because of the changes in the devices in the testbed, different results are generated with the results shown in the paper.

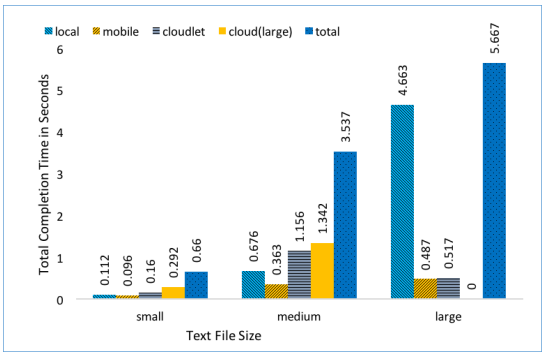
[9]https://github.com/mamoc-repos/MAMoC-iOS



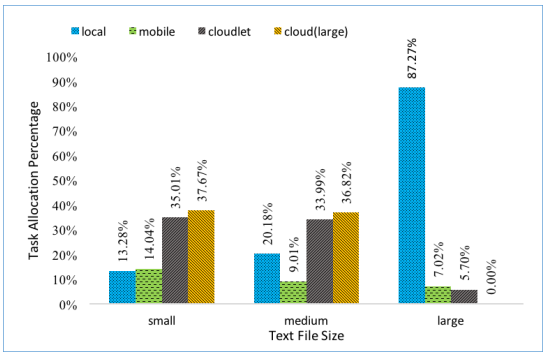
(a) Full offloading



(b) Partial offloading with decision engine



(c) Multisite offloading completion times



(d) Multisite task allocation percentage

Text file size	Offloadee	Full offloading	Partial offloading (equal task partitioning)	Partial offloading (single)			Partial offloading (multi)		
		Completion time	Completion time	Task partitioning percentage		Completion time	Task partitioning percentage		Completion time
Small	**Mobile**	0.714	0.811	48.60%	51.40%	0.775	13.28%	14.04%	0.659
	Cloudlet	0.458	0.683	27.90%	72.10%	0.544		35.01%	
	Cloud (small)	0.908	0.915	35.17%	64.83%	0.865		-	
	Cloud (medium)	0.859	0.898	28.57%	71.43%	0.811		-	
	Cloud (large)	0.808	0.865	26.45%	73.55%	0.785		37.67%	
Medium	**Mobile**	4.121	3.852	69.12%	30.88%	3.291	20.18%	9.01%	3.537
	Cloudlet	3.575	3.535	37.35%	62.65%	3.418		33.99%	
	Cloud (small)	5.498	4.507	55.25%	44.75%	4.370		-	
	Cloud (medium)	4.562	4.039	40.25%	59.75%	4.082		-	
	Cloud (large)	3.713	3.579	36.54%	63.46%	3.572		36.82%	
Large	**Mobile**	7.268	6.724	92.55%	7.45%	6.128	87.27%	7.02%	5.667
	Cloudlet	9.074	7.307	93.85%	6.15%	6.948		5.70%	
	Cloud (small)	11.033	8.074	100%	0%	5.376		-	
	Cloud (medium)	10.767	8.055	100%	0%	5.373		-	
	Cloud (large)	10.508	7.872	100%	0%	5.375		0%	

(e) The task partitioning evaluation results

Fig. 6.10 The reported results from [136]

6.4 MCDM Evaluations

This section provides a detailed MCDM evaluation based on the models developed in Section 4.5 in Chapter 4. As shown in the previous evaluation set of Section 6.2, the offloading site selection is performed by the AHP and fuzzy TOPSIS methodologies. Subsection 6.4.1 demonstrates the results of the single decision maker, which is used in the previous evaluation. Then, Subsection 6.4.2 demonstrates an example of group decision making based on the five DMs that were presented in Subsection 4.5.2.

Criteria	**Condition**	**Fuzzy value**
Bandwidth	$RTT < 20$	VH
	$20 < RTT < 50$	H
	$50 < RTT < 100$	G
	$100 < RTT < 200$	L
	$RTT > 200$	VL
Speed	$CP > (CP_{SN} * 3)$	VH
	$CP > (CP_{SN} * 2)$	H
	$CP > CP_{SN}$	G
	$CP == CP_{SN}$	L
	$CP < CP_{SN}$	VL
Availability	$MN : BL == 100$	VH
	$MN : 100 > BL > 80$	H
	$MN : 80 > BL > 50$	G
	$MN : 50 > BL > 20$	L
	$MN : 20 > BL > 0$	VL
	EN	H
	PN	VH
Security	MN	H
	EN	H
	PN	L
Price	MN	VL
	EN	L
	PN	VH

Table 6.4 Fuzzy value assignment based on criteria conditions of the offloading sites

6.4.1 Single Decision Making

The process of multi-criteria solver decision making was illustrated earlier in Section 4.5. It was also shown in Section 5.3, how the context profilers collect device and network contextual information to be fed to the offloading decision

engine used in real-world mobile applications. The DM provides expert knowledge for setting priorities among the different criteria affecting the offloading decision.

The fuzzy values for each criterion are assigned using the rules shown in Table 6.4. The lower the RTT values between the Host Mobile Device and the offloading site, the higher the fuzzy linguistic term would be for the site. The computation power of the site is compared to the host's computation power in terms of being more powerful in n-fold terms. The availability of the mobile nodes depends on the battery level (if the battery state is charging BL is set to 100 hence VH is assigned to the MN). The edge nodes are considered to be of high availability with public nodes being the highest in availability, as described in Subsection 2.1.4. Security and privacy can be interpreted differently according to the authentication mechanisms and the nature of connection establishment between the host node and the participating site. The security in MCO studies is one of the major concerns. According to [5], data is more secure when it is closer and more controllable when the source of computation is closer to the user. Therefore, higher security is given to both MNs and ENs while PNs are considered less secure as offloading destinations. The pricing and incentive models are active areas in MCC research as well. As mentioned in the Section 5.2, the local devices could belong to the same mobile user so the price is set to lowest for the devices in LAN. Conversely, the PNs has high monetary costs, so the price criterion is set to the highest for them.

The results of fuzzy TOPSIS analysis are summarized in Table 6.5. D_i^+ and D_i^- can be calculated using Eq. (4.26a) and Eq. (4.26b) respectively. Based on C_i^* values calculated by Eq. (4.28), the ranking of the sites in descending order is edge, mobile, and public as depicted in Table 6.6.

	Bandwidth	**Speed**	**Availability**	**Security**	**Price**
Mobile	VH	H	G	H	VL
Edge	H	VH	H	H	L
Public	G	VH	VH	L	VH
Mobile	(0.75, 1.0, 1.0)	(0.5, 0.75, 1.0)	(0.25, 0.5, 0.75)	(0.75, 1.0, 1.0)	(0.0, 0.0, 0.25)
Edge	(0.5, 0.75, 1.0)	(0.75, 1.0, 1.0)	(0.5, 0.75, 1.0)	(0.50, 0.75, 1.0)	(0.0, 0.25, 0.50)
Public	(0.25, 0.5, 0.75)	(0.75, 1.0, 1.0)	(0.75, 1.0, 1.0)	(0.0, 0.25, 0.5)	(0.75, 1.0, 1.0)
Weights	0.4073	0.3886	0.1084	0.0573	0.0385
Mobile	(0.305, 0.389, 0.389)	(0.194, 0.291, 0.389)	(0.097, 0.194, 0.291)	(0.194, 0.291, 0.389)	(0.0, 0.0, 0.097)
Edge	(0.204, 0.291, 0.389)	(0.291, 0.389, 0.389)	(0.194, 0.291, 0.389)	(0.194, 0.291, 0.389)	(0.0, 0.097, 0.194)
Public	(0.102, 0.194, 0.291)	(0.291, 0.389, 0.389)	(0.291, 0.389, 0.389)	(0.0, 0.097, 0.194)	(0.291, 0.389, 0.389)
S^+	$v_1^+ = (0.75, 1.0, 1.0)$	$v_2^+ = (0.75, 1.0, 1.0)$	$v_3^+ = (0.75, 1.0, 1.0)$	$v_4^+ = (0.75, 1.0, 1.0)$	$v_5^+ = (0.0, 0.0, 0.25)$
S^-	$v_1^- = (0.0, 0.0, 0.25)$	$v_2^- = (0.0, 0.0, 0.25)$	$v_3^- = (0.0, 0.0, 0.25)$	$v_4^- = (0.0, 0.0, 0.25)$	$v_5^- = (0.75, 1.0, 1.0)$

Table 6.5 Weighted fuzzy evaluation matrix for offloading sites

Offloading site	D_i^+	D_i^-	C_i^*
Edge	2.5123	1.7688	0.4132
Mobile	2.6316	1.7492	0.3993
Public	2.9675	1.3418	0.3114

Table 6.6 Fuzzy TOPSIS results (sorted by C_i^*) for the single decision maker

6.4.2 Group Decision Making

To evaluate the scalability of the proposed multi-criteria solver algorithm, a number of extra nodes are added to the experiment environment for the GDM scenario. To simulate real-life situations, the computational power, bandwidth rate, and battery level of the nodes will be varied. Similar to previous experiments, the Host Mobile Device is still the Nexus 7 which is identified as (Nexus 7-1) to extinguish it from the other two Nexus 7 devices in the testbed. Three nearby devices (Pixel, Nexus 7-2, Nexus 7-3) are added to the nearby device group to form MAC. Two new edge VMs are also spawned from the same school server that was used for the previous experiments. However, these two additional servers have a different number of CPUs and memory allocated to them. The same three public cloud instances that were used in Section 6.2 are also used for this test. Table 6.7 lists the hardware specifications and IDs of all the offloading sites.

Node	**CPU (in GHz)**	**RAM (in GB)**	**OS**	**ID**	**Variations**
Nexus 7-2	1.3 (Dual)	1	Android 6.0	Mobile-1	$BL = 85$
Nexus 7-3	1.3 (Dual)	1	Android 6.0	Mobile-2	$BL = 31$
Pixel	2.15 (Quad)	2	Android 9.0	Mobile-3	$BL = 62$
Edge (small)	2.4 (Dual)	4	Ubuntu 17.04	Edge-1	-
Edge (medium)	2.4 (Quad)	8	Ubuntu 17.04	Edge-2	-
Edge (large)	2.4 (Octa)	16	Ubuntu 17.04	Edge-3	-
Cloud (small)	2.4 (Single)	1	Amazon Linux 2	Public-1	eu-west-2 (London)
Cloud (medium)	2.8 (Quad)	7.5	Amazon Linux 2	Public-2	us-east-2 (Ohio)
Cloud (large)	2.8 (Octa)	15	Amazon Linux 2	Public-3	us-west-2 (Oregon)

Table 6.7 Experimental environment device specifications for the MCDM GDM evaluation

In order to show how the different DMs work in practice, we consider the five DMs proposed in Subsection 4.5.2 in Chapter 4. The criteria preference of all the five decision makers were shown in Table 4.5. The result of these decisions generates the following judgement matrices $A^{(1)}$,...,$A^{(5)}$ on a set of five criteria for evaluating the 9 alternatives (offloading sites). Let $w^{(k)} = (w1^{(k)},...,w5^{(k)})$ be the individual priority vector derived from judgment matrix $A^{(k)}$ using RGMM [34]. $A^{(k)}$ and $w^{(k)}$ (k = 1,2,...,5) are listed below.

$$A^{(1)} = \begin{pmatrix} 1 & 1 & 5 & 7 & 9 \\ 1 & 1 & 5 & 6 & 8 \\ 1/5 & 1/5 & 1 & 3 & 3 \\ 1/7 & 1/6 & 1/3 & 1 & 2 \\ 1/9 & 1/8 & 1/3 & 1/2 & 1 \end{pmatrix}, \quad w^{(1)} = \{0.4072, 0.3885, 0.1083, 0.0572, 0.0384\}^T$$

$$A^{(2)} = \begin{pmatrix} 1 & 5 & 7 & 9 & 9 \\ 1/5 & 1 & 3 & 7 & 7 \\ 1/5 & 1/5 & 1 & 3 & 3 \\ 1/9 & 1/7 & 1/5 & 1 & 1 \\ 1/9 & 1/7 & 1/5 & 1 & 1 \end{pmatrix}, \quad w^{(2)} = \{0.5888, 0.2219, 0.1177, 0.0357, 0.0357\}^T$$

$$A^{(3)} = \begin{pmatrix} 1 & 1/7 & 1/5 & 7 & 7 \\ 7 & 1 & 3 & 9 & 9 \\ 5 & 1/3 & 1 & 9 & 9 \\ 1/7 & 1/9 & 1/9 & 1 & 1 \\ 1/7 & 1/9 & 1/9 & 1 & 1 \end{pmatrix}, \quad w^{(3)} = \{0.1248, 0.5146, 0.2988, 0.0307, 0.0307\}^T$$

$$A^{(4)} = \begin{pmatrix} 1 & 1 & 1 & 1/9 & 3 \\ 1 & 1 & 3 & 1/9 & 3 \\ 1 & 1/3 & 1 & 1/9 & 3 \\ 9 & 9 & 9 & 1 & 9 \\ 1/3 & 1/3 & 1/3 & 1/9 & 1 \end{pmatrix}, \quad w^{(4)} = \{0.09, 0.1184, 0.0751, 0.6767, 0.0394\}^T$$

$$A^{(5)} = \begin{pmatrix} 1 & 1 & 3 & 3 & 1/9 \\ 1 & 1 & 3 & 3 & 1/9 \\ 1/3 & 1/3 & 1 & 3 & 1/9 \\ 1/3 & 1/3 & 1/3 & 1 & 1/9 \\ 9 & 9 & 9 & 9 & 1 \end{pmatrix}, \quad w^{(5)} = \{0.1125, 0.1125, 0.0615, 0.0391, 0.6741\}^T$$

The group priority vector $w^{(G)}$ and group judgement matrix $A^{(G)}$ are calculated according to Eq. (4.19) and Eq. (4.20) explained previously in Chapter 4.

$$A^{(G)} = \begin{pmatrix} 1 & 0.934 & 1.838 & 2.713 & 2.852 \\ 1.069 & 1 & 3.322 & 2.63 & 2.786 \\ 0.544 & 0.31 & 1 & 2.141 & 2.141 \\ 0.368 & 0.38 & 0.467 & 1 & 1.148 \\ 0.35 & 0.358 & 0.467 & 0.87 & 1 \end{pmatrix}, \quad w^{(G)} = \{0.1979, 0.2261, 0.112, 0.0699, 0.0646\}^T$$

Equal weights ($\pi = \pi_1, \pi_2, ..., \pi_5$) of 1.0/5.0 are assigned to each decision maker for calculating the group priority vector and the group judgment matrix. In particular situations, the application developer might wish to assign different weights to each DM according to application requirements, as described in Subsection 4.5.2.

	Bandwidth	Speed	Availability	Security	Price
Mobile-1	VH	L	H	H	VL
Mobile-2	VH	L	L	H	VL
Mobile-3	VH	G	G	H	VL
Edge-1	H	G	H	H	L
Edge-2	H	H	H	H	G
Edge-3	H	VH	H	H	H
Public-1	G	G	VH	L	G
Public-2	L	H	VH	L	H
Public-3	VL	VH	VH	L	VH
Mobile-1	(0.75, 1, 1)	(0, 0.25, 0.5)	(0.5, 0.75, 1)	(0.5, 0.75, 1)	(0, 0, 0.25)
Mobile-2	(0.75, 1, 1)	(0, 0.25, 0.5)	(0, 0.25, 0.5)	(0.5, 0.75, 1)	(0, 0, 0.25)
Mobile-3	(0.75, 1, 1)	(0.25, 0.5, 0.75)	(0.25, 0.5, 0.75)	(0.5, 0.75, 1)	(0, 0, 0.25)
Edge-1	(0.5, 0.75, 1)	(0.25, 0.5, 0.75)	(0.5, 0.75, 1)	(0.5, 0.75, 1)	(0, 0.25, 0.5)
Edge-2	(0.5, 0.5, 1)	(0.5, 0.75, 1)	(0.5, 0.75, 1)	(0.5, 0.75, 1)	(0, 0.25, 0.5)
Edge-3	(0.5, 0.75, 1)	(0.75, 1, 1)	(0.5, 0.75, 1)	(0.5, 0.75, 1)	(0, 0.25, 0.5)
Public-1	(0.25, 0.5, 0.75)	(0.25, 0.5, 0.75)	(0.75, 1, 1)	(0, 0.25, 0.5)	(0.25, 0.5, 0.75)
Public-2	(0, 0.25, 0.5)	(0.25, 0.5, 0.75)	(0.75, 1, 1)	(0, 0.25, 0.5)	(0.5, 0.75, 1)
Public-3	(0, 0, 0.25)	(0.75, 1, 1)	(0.75, 1, 1)	(0, 0.25, 0.5)	(0.75, 1, 1)

Table 6.8 The assigned fuzzy values for offloading sites: MCDM-GDM

The results of the weighted normalised fuzzy values and final site rankings are shown in Table 6.9 and Table 6.10 accordingly.

It can be observed from the site ranking results that the two nearby mobile devices which are Mobile-3 (Pixel) and Mobile-1 (Nexus 7-2) are ranked highly due to high speed and availability and low bandwidth between them and the Host Mobile Device (Nexus 7-1). Mobile-2 is ranked after the edge nodes due to its low battery level, which leads to lower availability than the other two mobile devices. The price models for the edge nodes are not as important as their speeds, therefore, the more computationally powerful edge nodes are ranked higher. However, this does not apply to the public nodes since the bandwidth of the more powerful public nodes is lower, resulting in lower rankings.

The results of each individual decision maker $DM^{(1)}$,...,$DM^{(5)}$ and final site rankings will be listed in Appendix B.

	Bandwidth	**Speed**	**Availability**	**Security**	**Price**
Weights	0.1979	0.2261	0.112	0.0699	0.0646
Mobile-1	(0.148, 0.226, 0.226)	(0.0, 0.057, 0.113)	(0.057, 0.113, 0.170)	(0.113, 0.170, 0.226)	(0.0, 0.0, 0.057)
Mobile-2	(0.148, 0.226, 0.226)	(0.0, 0.057, 0.113)	(0.0, 0.057, 0.113)	(0.113, 0.170, 0.226)	(0.0, 0.0, 0.057)
Mobile-3	(0.148, 0.226, 0.226)	(0.113, 0.170, 0.226)	(0.0, 0.057, 0.113)	(0.113, 0.170, 0.226)	(0.0, 0.0, 0.057)
Edge-1	(0.099, 0.170, 0.226)	(0.057, 0.113, 0.170)	(0.113, 0.170, 0.226)	(0.113, 0.170, 0.226)	(0.0, 0.057, 0.113)
Edge-2	(0.099, 0.170, 0.226)	(0.113, 0.170, 0.226)	(0.113, 0.170, 0.226)	(0.113, 0.170, 0.226)	(0.057, 0.113, 0.170)
Edge-3	(0.099, 0.170, 0.226)	(0.170, 0.226, 0.226)	(0.113, 0.170, 0.226)	(0.113, 0.170, 0.226)	(0.113, 0.170, 0.226)
Public-1	(0.099, 0.170, 0.226)	(0.0, 0.057, 0.113)	(0.170, 0.226, 0.226)	(0.0, 0.057, 0.113)	(0.057, 0.113, 0.170)
Public-2	(0.049, 0.113, 0.170)	(0.113, 0.170, 0.226)	(0.170, 0.226, 0.226)	(0.0, 0.057, 0.113)	(0.113, 0.170, 0.226)
Public-3	(0.0, 0.057, 0.113)	(0.170, 0.226, 0.226)	(0.170, 0.226, 0.226)	(0.0, 0.057, 0.113)	(0.170, 0.226, 0.226)

Table 6.9 Weighted fuzzy evaluation for group judgement matrix of offloading sites: MCDM-GDM

Offloading site	D_i^+	D_i^-	C_i^*
Mobile-3	3.1995	1.3835	0.3019
Mobile-1	3.2557	1.3515	0.2933
Edge-3	3.1515	1.3017	0.2923
Edge-2	3.0963	1.2774	0.2921
Edge-1	3.0920	1.2665	0.2906
Mobile-2	3.3120	1.3505	0.2897
Public-1	3.2850	1.2567	0.2767
Public-2	3.2588	1.2043	0.2698
Public-3	3.3221	1.2138	0.2676

Table 6.10 Final ranking of the offloading sites: MCDM-GDM

6.5 Application Refactoring Evaluation

In this section, the server component, which is responsible for parsing, decompiling, and recompiling APK files, is evaluated. A list of mobile applications and their application identifiers are first collected to be used in this evaluation process. The applications used for refactoring are the ones investigated by the researchers in [130]. Due to the unavailability of some links of the applications provided in the paper, alternative applications are used as shown in Table 6.11. The applications are widely used in conducting MCC studies by researchers [130].

This main goals of conducting this experiment are to explore the computational costs of refactoring mobile cloud applications and whether it is feasible in practice. The limitations of the refactoring process can also be explored through retargeting Android market applications, which will be discussed in Section 6.7. The answers to these questions will determine the degree to which this is a useful tool for extracting code for further analysis.

Application Name	**Application ID**
Sudoku game	org.moire.opensudoku
N-Queens game	com.memmiolab.queens
Gobang game	ric.ov.SimplyGomoku
Video downloader for Twitter	com.billApps.VTLoader
OpenCV face detection	com.ollieteam.facedetection
Photoshoot Game	com.appsdgl.photoPuzzle
DealsPure app	com.dealspure.wild
Opera Mini Browser	com.opera.mini.native
Bing Image Search	de.devmil.muzei.bingimage
Flickr Mobile	pl.eprogmedia.flickrmobile
Linpack	org.skynetsoftware.linpack
BBC News	bbc.mobile.news.ww
Smart News	jp.gocro.smartnews.android
Euro News	com.euronews.express
Twitter Lite	com.twitter.android.lite
Last.fm	com.adriannieto.lastfmtops
Chess Game	com.cnvcs.chess
Basic Physics	com.zayaninfotech.physics
Antivirus	com.antivirus.applock
Photoshop Viewer	com.psd.viewer
Applock	mobilesecurity.applockfree
Expense Manager	com.mlab.expense.manager

Table 6.11 Benchmarking applications used for application refactoring evaluation

6.5.1 Experimental Environment

The edge server identified as (Edge-3) in the previous experiment is used to conduct this test. MAMoC Server uses a Python library called *Androguard* to aid with the process of parsing and analysing APKs. Under the hoods, Androguard uses Jadx tool which was described in Section 5.6.2.2 as a tool used in MAMoC Client for decompiling the Java codes from the application bytecode in the Host Mobile Device. The steps of application refactoring were explained earlier in Section 5.7.2.4. The same steps are applied to each application in this experiment. The following metrics are collected in the process:

- **Total number of classes:** this counts every class which are produced from the decompilation process of the application.
- **Total number of methods:** each class contains a number of methods that are all aggregated into this field.
- **Filtered classes:** the *dx.get_classes()* method provided by AndroGuard returns a list of *ClassAnalysis* objects with some of them labelled as "EXTERNAL". This label indicates that the source code of the class is not defined within the DEX files that are loaded inside the analysis. For example, *java.io.FileNotFoundException* is an API class, and it is not included in the DEX files as it is available on the system.
- **Classes with code:** Some of the classes are resource or layout related classes that do not contain any Java code. Only the classes with code are filtered for further analysis.
- **Offloadable classes:** the filtered classes with codes are scanned for dependency on any native device features and Android-specific library calls. The classes that only contain pure Java code and can be executed on JVM are marked and annotated as @Offloadable.
- **Elapsed time:** this is the time taken from the start of the analysis until the offloadable classes are identified and annotated.

6.5.2 Results and Analysis

The application IDs are saved in a text file and passed to the application refactor component in MAMoC Server to be read. The script starts by downloading the APK and applying the steps described in Section 5.7.2.4 for each file. The

output is a text file that includes the metrics and a signed APK file to be distributed and installed to the mobile devices.

It can be observed from the refactoring results, the smaller apps with fewer classes and methods are processed in a shorter time. However, this pattern is not standard among all the apps, for instance, the Sudoku app with a lower number of classes took 16.95 seconds to be processed while the N-Queens app with a higher number of classes took 11.31 seconds. Meanwhile, the GoBang game which has less than half the number of classes compared with the Sudoku and N-Queens takes about 5 times longer to be processed.

For the larger apps such as Opera browser and BBC apps with over 20,000 classes, it can take up to or more than 10 minutes to produce the results which can be impractical in real-life scenarios.

Application ID	**Classes**	**Methods**	**Filtered classes**	**Classes with code**	**Offload--ables**	**Time (in sec)**
org.moire.opensudoku	1208	8795	401	394	100	16.95
com.memmiolab.queens	1486	11268	33	28	17	11.31
ric.ov.SimplyGomoku	620	3506	278	267	32	77.83
com.billApps.VTLoader	8238	51577	5894	5790	272	250.29
com.ollieteam.facedetection	5882	35010	4490	4453	18	197.16
com.appsdgl.photoPuzzle	3114	19977	1823	1789	30	130.08
com.dealspure.wild	2059	14502	548	535	52	21.89
com.opera.mini.native	20173	102465	18386	17972	5968	571.76
de.devmil.muzei.bingimage	3212	20574	1386	1302	34	58.03
pl.eprogmedia.flickrmobile	3419	26610	945	880	43	64.27
org.skynetsoftware.linpack	38	117	9	9	1	1.79
bbc.mobile.news.ww	23838	119000	20337	19983	4257	689.46
jp.gocro.smartnews.android	9947	62031	8213	7977	1130	302.01
com.euronews.express	9231	58811	7261	7038	1168	270.51
com.twitter.android.lite	2464	14169	1901	1859	303	85.91
com.adriannieto.lastfmtops	3670	22881	2021	1961	635	135.82
com.cnvcs.chess	2258	11163	1886	1853	3	125.1
com.zayaninfotech.physics	712	4352	217	206	13	11.56
com.antivirus.applock	11096	62213	8840	8641	804	341.17
com.psd.viewer	11149	65752	9735	9552	3215	375.04
mobilesecurity.applockfree	5207	28725	3424	3360	140	192.07
com.mlab.expense.manager	15564	101512	11788	11281	1163	537.27

Table 6.12 Application refactoring evaluation results

Table 6.12 shows the results of running the application refactor component in MAMoC Server for the selected benchmarking applications.

6.6 Evaluation of Requirements

This section briefly evaluates MAMoC design and reference implementation, presented in the previous chapters, against the requirements reported in Chapter 4. This is a descriptive evaluation and performed manually without any dynamic environmental set-ups to check whether/how the defined requirements are met.

1. **Functional Requirements**

   ✓ **Offloading compute-intensive tasks:** MAMoC supports offloading with the use of annotations from developers or automatically through application refactoring steps as evaluated in the previous section.

   ✓ **Supporting local and remote executions:** Both local and remote as well as local+remote executions were shown in the first two sets of evaluations in this chapter.

   ✓ **Discovering offloading sites:** The device-to-device and device-to-server communications are handled by the service discovery component which keeps track of the changes in the nearby devices with the broadcasting technique and in the servers with the Pub/Sub mechanisms.

   ✓ **Selecting the most optimal offloading site(s):** The choices of selecting sites are performed using both offloading score and MCDM methodologies. The evaluations in Section 6.3 and Section 6.2.

   ✓/× **Managing the offloading sites:** The developers can manage the offloading sites through the installed framework on the mobile devices and Docker containers in the servers. However, no management tools are available for normal users or system administrators to manage the sites through a user-friendly Graphical User Interface.

2. **Non-functional Requirements**

   ✓ **Performance enhancement:** The performance of the running tasks were improved in some of the offloading scenarios for both single-site and multisite offloading evaluations.

   ✓ **Energy efficiency:** Less energy is consumed from Host Mobile Device in some of the multisite offloading scenarios as shown in Section 6.2.

   ✓ **Simplicity and ease of deployment** Both MAMoC Client and MAMoC Server are made available through Gradle library and docker images respectively. The steps of integrating the client library to the

mobile applications are given in Section 5.8. Moreover, the installation of the server-side containers are provided in the respective Github repository.

✓/× **Framework reusability and extensibility:** The modularity approach in developing the framework allows other developers to add extra modules to the system. Nonetheless, there are no clear guidelines and no instructions are provided for extending the framework with extra features.

✓/× **Fault tolerance and reliability:** As it will be later discussed in Section 6.7, an offloading request timeout mechanism is used to recover from failures. Even though the task execution is gradually performed, a more robust failure recovery mechanism can improve the reliability of the system.

✓/× **Security and privacy:** Subsection 5.4.3 discussed the two approaches used for securing both D2D and device-to-server communications. However, there are other security and privacy issues with MCO systems which are discussed in detail in [5].

6.7 Discussion and Limitations

The evaluation results of the first two experiments showed that in particular instances, multisite offloading can both reduce the overall completion time and energy consumption of running the tasks. Despite its advantages, the following limitations can be observed across both evaluations:

- The performed evaluation in a controlled and well-defined environment while performing different offloading scenarios. In more realistic scenarios, mobile users move around and the network signal strength changes, which affect the changes in RTT values. Moreover, the use of a strong signal Wi-Fi (CS network) for this evaluation clearly improves the results of the offloading, especially to the cloudlet and cloud nodes. Figuratively, using cellular networks would increase the RTT values of the offloading sites that result in lower task partitions allocated to them. A more dynamic network and updated RTT values can be used to showcase the differences in the results for better testing the decision engine.

- One more concern in the evaluations is the occurrence of offloading failures. With intermittent connectivities between the Self Node and offloading sites,

communication failures happen. This evaluation uses a timeout mechanism of 5 seconds to fall back the execution to the local device. For a more robust offloading system, a more intelligent failure recovery mechanism can be adapted similar to the works discussed in [163].

- Despite its benefits, our approach is not applicable to all applications. Some mobile applications are written in a monolithic style, in which functionality cross-cuts through traditional modularization program constructs such as classes and methods. Without clear offloading program points, our approach would be inapplicable.

- MAMoC only considers a single mobile user environment without considering other mobile users which increase the number of offloaded requests and adds more request loads on the service providers. The application is running a single-user environment so that the execution time for each offloading of the same task on the same computation node is generally close to their average. This cannot be guaranteed for a multi-user environment. Researchers in [20] use game-theoretic approaches to model this scenario and handle the load balancing in the sites. In addition, a few related studies can be introduced to enhance MAMoC, such as supporting multi-user cases via game-theoretic model [25] and supporting complex mobility models via other offloading decision algorithms [125].

- The complex mobility model of mobile devices is a remaining challenge. The network conditions between the mobile device and the same offloading site in the same location are usually close to their average. In a real-world environment, although the performance improvement may be marginally different.

- Regarding the energy consumption readings, not all devices support the Trepn profiler [98]. Alternative software or hardware power consumption measurement tools need to be explored for the modern devices.

AHP is easy to use, scalable, and the hierarchy structure can easily adjust to fit many sized problems. TOPSIS has a simple process, easy to use and program, and scalable in how the number of steps remains the same regardless of the number of alternatives (offloading sites).

The evaluation showed a simple and flexible method for generating the rankings of the nodes. It also demonstrated how DMs can analyze the elasticity of the final decision by applying the sensitivity weights to them. It is also possible to

measure the consistency of a decision maker's judgements and refine their pairwise comparisons to reach a consistency below the threshold. However, as any other decision-making mechanisms, MCDM has its own limitations:

- Subjectivity is one of the major concerns of MCDM approaches. To reduce subjectivity in decision making one can use hybrid approaches comprising the method this work uses by introducing fuzzy range values as shown by the work in [165].
- Generality: nearby mobile devices are generally considered more secure than remote devices for processing and storing data [95]. However, there might be situations where nearby mobile devices cannot be trusted or a fake edge node are installed in the local network.
- Regarding the group decision making, a consensus needs to be reached between all the DMs to generate an aggregated decision matrix for evaluating and ranking the nodes. An application developer might not understand this and assign unrealistic pairwise comparison values that violate the judgements of other decision makers and disrupt the evaluation process.

The application refactoring evaluation demonstrated that the proposed utility can successfully parse and decompile APKs, annotate the offloadable classes and generate a signed copy of the APK file. It was also observed from the results that the elapsed time of the process was non-deterministic. Even though the tool was shown to generate results for the benchmarking apps, there are a number of limitations explained below:

- This utility requires a sizable amount of memory in order to perform classification when operating in package mode. All the benchmarked applications are of small size APKs (less than 10 MB in size). For the bigger applications, multiple errors occurred which disrupted the decompilation process. For instance, all the top 10 applications on the Play Store were not capable of being used in the tool because of their large sizes.
- The Dex2Jar utility is considered being able to decompile the obfuscated applications which are encrypted by the developers before releasing them to the Play Store. In the evaluation, most of the classes were produced correctly but not all of them are investigated due to some high number of offloadable classes in some application refactoring results.

- Not all Android market applications are written in native Java. There are many modern applications written in Kotlin, hybrid applications written in Javascript with user interfaces built with HTML/CSS and games written in Android native structure using C++ or Unity engine. This cannot be easily discovered before starting the decompilation process and scanning the source files.

6.8 Summary

In this chapter, the proposed MAMoC framework is evaluated in real-world environments. Four sets of experiments are conducted to showcase the research hypotheses of this thesis. The offloading decision algorithm evaluation used three different tasks for the completion time and energy consumption and compared the results of full single-site offloading and our proposed multisite offloading as well as comparing it with an offloading framework in the literature. The task partitioning evaluation demonstrated running a data-intensive task in different offloading scenarios and the benefits of multisite offloading for the distributed subtasks. The MCDM evaluation expanded on the multi-criteria solver component by testing the group decision making concept. Finally, the application refactor evaluation presented the benchmarked applications and results of the decompilation process. The evaluation outcomes are then discussed with identifying the shortcomings and limitation in conducting them.

Chapter 7

Conclusion and Future Work

7.1 Summary of Thesis

Above all, as a solution to the problem of the limited battery capacity of the mobile device, computation offloading in mobile devices can be used to improve performances and save the energy of mobile devices. In order to fully utilise the computing and battery capacity of the idle or light load devices, an end-to-end mobile computation offloading framework has been proposed to enhance the performance and minimise the energy consumption in mobile devices while guaranteeing the performance requirements of the offloaded tasks.

- Chapter 1 presented the challenges facing the lower end mobile devices and motivations for conducting this study. It also listed the research hypotheses that this work is investigating and the objectives of solving the relevant research problems.
- Chapter 2 provided an overview of the most common mobile cloud architectures, including MCC, MEC, and MFC. It also described the research interest in MCO and the two central aspects of it that derive the body of this work.
- Chapter 3 surveyed the existing work in the multisite MCO works and derived a taxonomy reflecting different aspects of the research in this area and listed current trends and future directions.
- Chapter 4 modelled the tasks of a mobile application in the Host Mobile Device and the heterogeneous computing nodes that act as offloading sites in this work. The offloading cost in terms of execution and energy were analysed. It then presented the offloading policy for partitioning parallalizable tasks

using the offloading scores of the sites. It then presented the multi-criteria solving algorithms based on AHP and fuzzy TOPSIS methods for evaluating the different offloading criteria and ranking the multiple available offloading sites.

- Chapter 5 presented the design and implementation of the framework which incorporated all the research presented in the previous chapter. The framework consisted of two main systems: MAMoC Client and MAMoC Server with each containing different loosely coupled services that communicated with each other.

  Section 5.6, Section 5.7 presented the implementation details of both MAMoC Client and MAMoC Server components accordingly. In MAMoC client, the service discovery discussed both communications with the nearby mobile devices in the form of D2D and other offloading sites with MAMoC router. The process of code decompilation and context profiling were also presented for preparing the offloading decision making. Finally, the deployment controller discussed the different deployment scenarios for both local and remote executions. In MAMoC server, the implementation details of the three modules running on separate containers including MAMoC router, server manager, and MAMoC repository were explained with diagrams and code listings.

- Chapter 6 presented a detailed evaluation of the research presented in this thesis. Four sets of experiments were conducted in this chapter. The first evaluated the performance of the task partitioning algorithm based on offloading scores of the nodes. The second evaluated the performance of the offloading decision algorithm based on the checkpoint variables and MCDM. The third added the GDM method to the experiment and presented the results of the aggregated decision makers. Finally, the fourth experiment evaluated the application refactoring component performance in parsing and decompiling unmodified APKs. The chapter also included a discussion of the results and a set of limitations of the evaluations.

7.2 Review of Hypotheses

This thesis has argued that an adaptive with multiple destinations mobile offloading workflow can be realised using offloading score approach for dividing the parallelised tasks into subtasks. The other approach is by using MCDM that evaluates different criteria to determine the most appropriate offloading sites that can execute these

tasks efficiently. An architectural system design and a reference implementation have been proposed to enable these approaches. The hypotheses presented in Chapter 1 can be evaluated by conducting experiments that attempt to confirm it, and by analysing the experimental results with the predicted consequences of the hypotheses.

H1 **Mobile devices can be seamlessly leveraged with all the surrounding sites including nearby mobile devices and edge servers as well as distant cloud resources. Moreover, the decision to offload a task and identify the most optimal candidate for single-site offloading and the most optimal ranking of the candidates for multisite offloading scenarios can be taken adaptively.**

Leveraging mobile devices with multiple service providers is the central theme of this work in the form of multisite offloading. This is achieved with the help of a service discovery that can be connected to multiple external resources simultaneously. Throughout Chapter 5, different components are discussed to achieve this goal. Adaptively choosing the optimal candidates has been discussed in Chapter 4. The adaptation to the dynamic changes in the mobile environment can be observed in the results of both single decision making and group decision making evaluations presented in Section 6.4.

H2 **The process of deploying runtime environments for serving offloading requests from mobile devices can be simplified. It can also be utilised to automatically identify offloadable tasks in unmodified mobile applications when manual task annotation is infeasible.**

The deployment of runtime environments are discussed in Section 5.7 and demonstrated in Subsection 6.1.1. The choice of lightweight containers has simplified the process of deploying mobile service providers. Compared to other deployment strategies in the literature [155] [154], the size of the containers and boot-up times are greatly reduced. Identifying the heavy parts of an application is considered one of the most challenging parts of designing mobile offloading systems since manual annotation introduces extra burden and work for developers [97]. In this work, a number of static analysis and refactoring approaches are used to generate the desired tasks. However, as it can be observed in the results of the evaluation in Section 6.5, the elapsed time and number of produced tasks are non-deterministic and further research needs to be performed to formalise this process.

7.3 Review of Contributions

This section revisits the contributions presented in Chapter 1 and discusses how they are achieved in this work with references to relevant chapters and sections in this thesis. This thesis provides a set of contributions to the body of research knowledge in the area of MCC generally, and multisite MCO specifically. These contributions include the following:

1. **Literature review:** this thesis provided an overview of the different mobile cloud architectures with a detailed comparison between them. It also analysed the existing multisite MCO solutions in the literature in which a taxonomy was derived for concluding the current trends and future directions.

2. **An adaptive task partitioning algorithm:** this algorithm was presented in Subsection 4.4.2 that decided on the task allocation percentages and execution locations for each subtask. The contextual changes in the dynamic mobile environment are profiled and reflected in the task partitioning and offloading decisions.

3. **An offloading site ranking mechanism:** the multi-criteria solver algorithm used two methods of MCDM and fuzzy logic to evaluate the criteria and rank the available offloading sites. Different optimization goals using MCDM group decision-making concept is also used for application-specific criteria priority requirements.

4. **Design and implementation of MAMoC:** this thesis defined a set of requirements for designing a robust mobile cloud offloading system. Distinctive design choices are made for making the components of the system both lightweight and easy to adapt. With keeping mobile application developers in mind, MAMoC Client is designed to work as a client library to be easily integrated with Android applications. It allows developers to use it as a simple programming model to build mobile cloud applications and abstract complex underlying heterogeneous technologies. MAMoC Server is a set of loosely coupled containers that work as a lightweight runtime environment for serving the mobile offloading requests on the heterogeneous offloading sites for providing a more scalable and reliable offloading service.

5. **Automated annotation generation:** this work also supports the arbitrary applications that are not specifically annotated by developers. It was shown in the steps demonstrated in Section 5.7.2.4 and evaluated in

Section 6.5 that it is possible to identify the offloadable classes and annotate them for use with MAMoC framework.

7.4 Future Works

There are some practical and theoretical issues that need to be addressed in the future to evolve MAMoC from a research tool to a platform for industrial and general-purpose use. These are research goals that were not met because of either time constraints or due to being out with the focus of this thesis.

- **Multi-user mobile offloading:** the proposed offloading decision algorithm makes decisions from a single view of Host Mobile Device. Developing algorithms to enable the framework to deal with user mobility and multi-user competition, including reducing the impact on low-power users is an interesting future direction to be considered. With the development of 5G, Artificial Intelligence, and other technologies, it is inevitable to use multiple mobile phones to enhance and change daily lifestyles. For example, a group of mobile phones can be used to process and filter all kinds of data from wearable devices for health data collection safely and efficiently.

- **Offloading failure recovery:** failures happen when the offloading request is not successfully executed on the offloading site or the necessary data is not wholly transmitted due to the intermittent wireless connection between the mobile device and the site. In its current form, MAMoC uses a timeout technique to fall back the execution to the local device. Future research works can investigate methods to handle failures during execution and more dynamic approaches to recover from these failures.

- **Evaluating real-world applications:** it would be valuable to test real-world applications such as face recognition or car license plate recognition apps on MAMoC framework. In evaluating the framework, we only used primitive benchmark tasks. In order to test the applicability of the framework, it needs to be tested on more complicated and real-world mobile applications. The guidelines on how to integrate existing or new mobile applications to the framework are provided in 5.8. It would also be beneficial if the evaluations are performed in real networks rather than in a laboratory network with real network conditions. The results can be compared and relevant further research can be performed.

- **Scheduling offloading requests:** MAMoC router forwards the offloading requests in a round robin fashion to the available servers. A load balancing consideration is needed to provide an overall better performance for the task scheduling strategy in the server-side. A possible approach is to use the recently popular Kubernetes technology stack for managing the server modules where the master node can work as MAMoC router and the pods on worker nodes to work as server managers.

- **AI-driven code transformation:** the code transformation component in MAMoC Server is manually written to transform Android-based Java code to pure Java code that can be compiled and executed on a general JVM. An AI-driven approach can be used to automate some transformation rules and adapt it for future transformations.

- **Eviction policy:** the offloading decision is dependent on the data stored from past executions. The current implementation does not foresee any eviction policy for the stored data. It means that the data may grow too big for the device itself. An optimal eviction policy must be employed in the database adapter to avoid reaching this point.

References

[1] Abbas, N., Zhang, Y., Taherkordi, A., and Skeie, T. (2018). Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465.

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

[3] Aguaron, J. and Moreno Jimnez, J. M. (2003). The geometric consistency index: Approximated thresholds. *European journal of operational research*, 147(1):137–145.

[4] Akherfi, K., Gerndt, M., and Harroud, H. (2018). Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics*, 14(1):1–16.

[5] Al-Mutawa, M. and Mishra, S. (2014). Data partitioning: An approach to preserving data privacy in computation offload in pervasive computing systems. In *Proceedings of the 10th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Q2SWinet '14, pages 51–60.

[6] Android version markets (2019). Android version market share distribution among smartphone owners as of september 2019. [online] https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/, last accessed on 18-10-2019.

[7] Android-x86 (2019). Android-x86 - porting android to x86. [online] https://www.android-x86.org/, last accessed on 23-09-2019.

[8] App downloads in Q2 of 2019 (2019). Q2 2019 was the biggest quarter for mobile to date. [online] https://www.appannie.com/en/insights/market-data/q2-2019-mobile-market-index/, last accessed on 06-09-2019.

[9] atteo/classindex (2019). Github. [Online] https://github.com/atteo/classindex, last accessed on 13-10-2019.

[10] Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., and Yang, H.-I. (2002). The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 87–92.

[11] Bangui, H., Ge, M., Buhnova, B., Rakrak, S., Raghay, S., and Pitner, T. (2017). Multi-criteria decision analysis methods in the mobile cloud offloading paradigm. *Journal of Sensor and Actuator Networks*, 6(4):25.

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

[13] Berg, F., Dürr, F., and Rothermel, K. (2016). Increasing the efficiency of code offloading in n-tier environments with code bubbling. In *Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 170–179.

[14] Bhardwaj, K., Sreepathy, S., Gavrilovska, A., and Schwan, K. (2014). Ecc: Edge cloud composites. In *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 38–47.

[15] Bhattacharya, A. and De, P. (2017). A survey of adaptation techniques in computation offloading. *Journal of Network and Computer Applications*, 78:97–115.

[16] Biswas, A. and Fujimoto, R. (2016). Profiling energy consumption in distributed simulations. In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '16, pages 201–209.

[17] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16.

[18] Budgen, D. and Brereton, P. (2006). Performing systematic literature reviews in software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 1051–1052.

[19] Camps-Mur, D., Garcia-Saavedra, A., and Serrano, P. (2013). Device-to-device communications with wi-fi direct: overview and experimentation. *IEEE wireless communications*, 20(3):96–104.

[20] Cardellini, V., Personé, V. D. N., Di Valerio, V., Facchinei, F., Grassi, V., Presti, F. L., and Piccialli, V. (2016). A game-theoretic approach to computation offloading in mobile cloud computing. *Mathematical Programming*, 157(2):421–449.

[21] Chen, C.-T. (2000). Extensions of the topsis for group decision-making under fuzzy environment. *Fuzzy sets and systems*, 114(1):1–9.

[22] Chen, E. Y. and Itoh, M. (2010). Virtual smartphone over ip. In *2010 IEEE International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 1–6.

[23] Chen, X., Chen, S., Zeng, X., Zheng, X., Zhang, Y., and Rong, C. (2017). Framework for context-aware computation offloading in mobile cloud computing. *Journal of Cloud Computing*, 6(1):1.

[24] Chen, X., Chen, Y., Ma, Z., and Fernandes, F. C. (2013). How is energy consumed in smartphone display applications? In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, pages 1–6.

[25] Chen, X., Jiao, L., Li, W., and Fu, X. (2015a). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808.

[26] Chen, Z., Jiang, L., Hu, W., Ha, K., Amos, B., Pillai, P., Hauptmann, A., and Satyanarayanan, M. (2015b). Early implementation experience with wearable cognitive assistance applications. In *Proceedings of the 2015 Workshop on Wearable Systems and Applications*, WearSys '15, pages 33–38.

[27] Cheng, Z., Li, P., Wang, J., and Guo, S. (2015). Just-in-time code offloading for wearable computing. *IEEE Transactions on Emerging Topics in Computing*, 3(1):74–83.

[28] Cheshire, S. and Krochmal, M. (2013a). Multicast dns. Technical report, IETF.

[29] Cheshire, S. and Krochmal, M. (2013b). Rfc 6763: Dns-based service discovery. Technical report, IETF.

[30] Cheshire, S. and Steinberg, D. (2006). *Zero Configuration Networking: The Definitive Guide*. Definitive Guide Series. O'Reilly Media.

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

[32] Cisco Visual Networking Index (2019). Cisco visual networking index: Forecast and trends, 2017–2022. [online] https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html, last accessed on 17-08-2019.

[33] Costamagna, V. and Zheng, C. (2016). Artdroid: A virtual-method hooking framework on android art runtime. In *IMPS@ ESSoS*, pages 20–28.

[34] Crawford, G. and Williams, C. (1985). A note on the analysis of subjective judgment matrices. *Journal of mathematical psychology*, 29(4):387–405.

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

[36] Desnos, A. and Gueguen, G. (2011). Android: From reversing to decompilation. *Blackhat*.

[37] Devos, M., Ometov, A., Mäkitalo, N., Aaltonen, T., Andreev, S., and Koucheryavy, Y. (2016). D2d communications for mobile devices: Technology overview and prototype implementation. In *2016 8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 124–129.

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

[39] Dong, Y., Zhang, G., Hong, W.-C., and Xu, Y. (2010). Consensus models for ahp group decision making under row geometric mean prioritization method. *Decision Support Systems*, 49(3):281 – 289.

[40] Drolia, U., Martins, R., Tan, J., Chheda, A., Sanghavi, M., Gandhi, R., and Narasimhan, P. (2013). The case for mobile edge-clouds. In *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, pages 209–215.

[41] Enzai, N. I. M. and Tang, M. (2016). A heuristic algorithm for multi-site computation offloading in mobile cloud computing. *Procedia Computer Science*, 80:1232–1241.

[42] Farrugia, S. (2016). Mobile cloud computing techniques for extending computation and resources in mobile devices. In *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 1–10.

[43] Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172.

[44] Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile cloud computing: A survey. *Future generation computer systems*, 29(1):84–106.

[45] Fernando, N., Loke, S. W., and Rahayu, W. (2016). Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds. *IEEE Transactions on Cloud Computing*.

[46] Flores, H., Hui, P., Nurmi, P., Lagerspetz, E., Tarkoma, S., Manner, J., Kostakos, V., Li, Y., and Su, X. (2018). Evidence-aware mobile computational offloading. *IEEE Transactions on Mobile Computing*, 17(8):1834–1850.

[47] Forman, E. and Peniwati, K. (1998). Aggregating individual judgments and priorities with the analytic hierarchy process. *European journal of operational research*, 108(1):165–169.

[48] Gai, K., Qiu, M., Zhao, H., Tao, L., and Zong, Z. (2016). Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *Journal of Network and Computer Applications*, 59:46–54.

[49] Gao, Y., Hu, W., Ha, K., Amos, B., Pillai, P., and Satyanarayanan, M. (2015). Are cloudlets necessary? *School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-15-139*.

[50] Giurgiu, I., Riva, O., Juric, D., Krivulev, I., and Alonso, G. (2009). Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In *Proceedings of the ACM/IFIP/USENIX 10th International Conference on Middleware*, Middleware'09, pages 83–102.

[51] Global-Micro-Server-Market-Forecasts-2024-Edge (2019). Global micro server market forecasts to 2024: Edge computing. [online] https://www.businesswire.com/news/home/20190715005393/en/Global-Micro-Server-Market-Forecasts-2024-Edge, last accessed on 22-07-2019.

[52] Google (2018). Profile battery usage with batterystats and battery historian. [online]https://developer.android.com/studio/profile/battery-historian, last accessed on 21-07-2019.

[53] Goudarzi, M., Zamani, M., and Haghighat, A. T. (2017). A fast hybrid multi-site computation offloading for mobile cloud computing. *Journal of Network and Computer Applications*, 80:219–231.

[54] Ha, K., Pillai, P., Richter, W., Abe, Y., and Satyanarayanan, M. (2013). Just-in-time provisioning for cyber foraging. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 153–166.

[55] Hao, S., Li, D., Halfond, W. G., and Govindan, R. (2013). Estimating mobile application energy consumption using program analysis. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 92–101.

[56] Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., and Koldehofe, B. (2013). Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pages 15–20.

[57] Hoque, M. A., Siekkinen, M., Khan, K. N., Xiao, Y., and Tarkoma, S. (2016). Modeling, profiling, and debugging the energy consumption of mobile devices. *ACM Computing Surveys (CSUR)*, 48(3):39.

[58] Hu, P., Dhelim, S., Ning, H., and Qiu, T. (2017). Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98:27–42.

[59] Hu, W., Gao, Y., Ha, K., Wang, J., Amos, B., Chen, Z., Pillai, P., and Satyanarayanan, M. (2016). Quantifying the impact of edge computing on mobile applications. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '16, pages 5:1–5:8.

[60] Hu, Y. C., Patel, M., Sabella, D., Sprecher, N., and Young, V. (2015). Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16.

[61] Huang, J., Qian, F., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. (2012). A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 225–238.

[62] Huerta-Canepa, G. and Lee, D. (2010). A virtual cloud computing provider for mobile devices. In *proceedings of the 1st ACM workshop on mobile cloud computing & services: social networks and beyond*, page 6.

[63] Jahanshahloo, G. R., Lotfi, F. H., and Izadikhah, M. (2006). Extension of the topsis method for decision-making problems with fuzzy data. *Applied Mathematics and Computation*, 181(2):1544–1551.

[64] Jararweh, Y., Doulat, A., AlQudah, O., Ahmed, E., Al-Ayyoub, M., and Benkhelifa, E. (2016). The future of mobile cloud computing: Integrating cloudlets and mobile edge computing. In *2016 23rd International conference on telecommunications (ICT)*, pages 1–5.

[65] Jin, X., Liu, Y., Fan, W., Wu, F., and Tang, B. (2017). Multisite computation offloading in dynamic mobile cloud environments. *Science China Information Sciences*, 60(8):89301.

[66] Joh, H. and Ryoo, I. (2015). A hybrid wi-fi p2p with bluetooth low energy for optimizing smart device's communication property. *Peer-to-Peer Networking and Applications*, 8(4):567–577.

[67] Joy, P. T. and Jacob, K. P. (2013). Cooperative caching framework for mobile cloud computing. *arXiv preprint arXiv:1307.7563*.

[68] Katoh, K., Misawa, K., Kuma, K.-i., and Miyata, T. (2002). Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic acids research*, 30(14):3059–3066.

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

[70] Khan, M. A., Cherif, W., Filali, F., and Hamila, R. (2017). Wi-fi direct research-current status and future perspectives. *Journal of Network and Computer Applications*, 93:245–258.

[71] Klauck, R. and Kirsche, M. (2012). Bonjour contiki: A case study of a dns-based discovery service for the internet of things. In *International Conference on Ad-Hoc Networks and Wireless*, pages 316–329.

[72] Knuth, D. E., Morris, Jr, J. H., and Pratt, V. R. (1977). Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350.

[73] Kosta, S., Aucinas, A., Pan Hui, Mortier, R., and Xinwen Zhang (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *2012 Proceedings IEEE INFOCOM*, pages 945–953.

[74] Kovachev, D., Yu, T., and Klamma, R. (2012). Adaptive computation offloading from mobile devices into the cloud. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 784–791.

[75] Kristensen, M. D. (2010). Scavenger: Transparent development of efficient cyber foraging applications. In *2010 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 217–226.

[76] Kumar, K., Liu, J., Lu, Y.-H., and Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140.

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

[78] Lai, Y.-J., Liu, T.-Y., and Hwang, C.-L. (1994). Topsis for modm. *European journal of operational research*, 76(3):486–500.

[79] Lee, J.-S., Su, Y.-W., Shen, C.-C., et al. (2007). A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi. *Industrial electronics society*, 5:46–51.

[80] Lewis, G. A., Lago, P., and Procaccianti, G. (2014). Architecture strategies for cyber-foraging: Preliminary results from a systematic literature review. In *European Conference on Software Architecture*, pages 154–169. Springer.

[81] Li, C., Xue, Y., Wang, J., Zhang, W., and Li, T. (2018). Edge-oriented computing paradigms: A survey on architecture design and system management. *ACM Computing Surveys (CSUR)*, 51(2):39.

[82] Li, J., Bu, K., Liu, X., and Xiao, B. (2013a). Enda: Embracing network inconsistency for dynamic application offloading in mobile cloud computing. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pages 39–44. ACM.

[83] Li, S., Farooqui, N., and Yalamanchili, S. (2013b). Software reliability enhancements for gpu applications. In *Sixth Workshop on Programmability Issues for Heterogeneous Multicores (MULTIPROG-2013)*.

[84] Li, Z., Wang, C., and Xu, R. (2001). Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246.

[85] Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., and Qureshi, A. (2015). Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, 48:99–117.

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

[87] Mahmud, R., Kotagiri, R., and Buyya, R. (2018). Fog computing: A taxonomy, survey and future directions. In *Internet of everything*, pages 103–130. Springer.

[88] Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358.

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

[90] Marinelli, E. E. (2009). Hyrax: cloud computing on mobile devices using mapreduce. Technical report, Carnegie-mellon univ Pittsburgh PA school of computer science.

[91] McGilvary, G. A., Barker, A., and Atkinson, M. (2015). Ad hoc cloud computing. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 1063–1068.

[92] Mobile Internet Usage (2015). Mobile internet usage skyrockets in past 4 years to overtake desktop as most used digital platform. [online] http://bit.ly/mobilemarket2015, last accessed on 13-06-2019.

[93] Mohammad, A.-R., Elham, A.-S., and Jararweh, Y. (2015). Amcc: Ad-hoc based mobile cloud computing modeling. *Procedia Computer Science*, 56:580–585.

[94] Monsoon (2018). High voltage power monitor. [online]https://www.msoon.com/high-voltage-power-monitor, last accessed on 21-07-2019.

[95] Mtibaa, A., Fahim, A., Harras, K. A., and Ammar, M. H. (2013). Towards resource sharing in mobile device clouds: Power balancing across mobile devices. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 51–56.

[96] Neto, J. L. D., Yu, S.-Y., Macedo, D. F., Nogueira, J. M. S., Langar, R., and Secci, S. (2018a). Uloof: a user level online offloading framework for mobile edge computing. *IEEE Transactions on Mobile Computing*, 17(11):2660–2674.

[97] Neto, J. L. D., Yu, S.-Y., Macedo, D. F., Nogueira, J. M. S., Langar, R., and Secci, S. (2018b). Uloof: a user level online offloading framework for mobile edge computing. *IEEE Transactions on Mobile Computing*, 17(11):2660–2674.

[98] Network, Q. D. (2018). Trepn power profiler. [online]https://developer.qualcomm.com/software/trepn-power-profiler, last accessed on 21-07-2019.

[99] Niu, R., Song, W., and Liu, Y. (2013). An energy-efficient multisite offloading algorithm for mobile devices. *International Journal of Distributed Sensor Networks*, 9(3):518518.

[100] Nucci, D. D., Palomba, F., Prota, A., Panichella, A., Zaidman, A., and Lucia, A. D. (2017). Software-based energy profiling of android apps: Simple, efficient and reliable? In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 103–114.

[101] Oberstein, T. and Goedde, A. (2015). The web application messaging protocol. *IETF working draft*.

[102] Octeau, D., Jha, S., and McDaniel, P. (2012). Retargeting android applications to java bytecode. In *Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering*, page 6.

[103] Orsini, G., Bade, D., and Lamersdorf, W. (2015). Context-aware computation offloading for mobile cloud computing: Requirements analysis, survey and design guideline. *Procedia Computer Science*, 56:10–17.

[104] Osdn.net (2019). Android-x86 release 9.0. https://osdn.net/projects/android-x86/releases/71931, last accessed on 09-12-2019.

[105] Ou, S., Yang, K., and Zhang, J. (2007). An effective offloading middleware for pervasive services on mobile devices. *Pervasive and Mobile Computing*, 3(4):362–385.

[106] Pathak, A., Hu, Y. C., and Zhang, M. (2012). Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 29–42.

[107] Qian, F., Wang, Z., Gerber, A., Mao, Z., Sen, S., and Spatscheck, O. (2011). Profiling resource usage for mobile applications: A cross-layer approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 321–334.

[108] Ra, M.-R., Sheth, A., Mummert, L., Pillai, P., Wetherall, D., and Govindan, R. (2011). Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 43–56.

[109] Rahimi, M. R., Venkatasubramanian, N., Mehrotra, S., and Vasilakos, A. V. (2012). 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*, UCC '12, pages 83–90, Washington, DC, USA. IEEE Computer Society.

[110] Rahimi, M. R., Venkatasubramanian, N., and Vasilakos, A. V. (2013). Music: Mobility-aware optimal service allocation in mobile cloud computing. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 75–82. IEEE.

[111] Raschka, S. (2014). About feature scaling and normalization (and the effect of standardization for machine learning algorithms). *Polar Political & Legal Anthropology Review*, 30(1):67–89.

[112] Ravi, A. and Peddoju, S. K. (2015). Handoff strategy for improving energy efficiency and cloud service availability for mobile devices. *Wireless Personal Communications*, 81(1):101–132.

[113] Recordon, D. and Reed, D. (2006). Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, pages 11–16.

[114] Rudenko, A., Reiher, P., Popek, G. J., and Kuenning, G. H. (1998). Saving portable computer battery power through remote process execution. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2(1):19–26.

[115] Saaty, T. L. (1989). Group decision making and the ahp. In *The analytic hierarchy process*, pages 59–67. Springer.

[116] Saaty, T. L. (2005). Analytic hierarchy process. *Encyclopedia of Biostatistics*, 1.

[117] Sanaei, Z., Abolfazli, S., Gani, A., and Buyya, R. (2014). Heterogeneity in mobile cloud computing: taxonomy and open challenges. *IEEE Communications Surveys & Tutorials*, 16(1):369–392.

[118] Sarkar, S., Chatterjee, S., and Misra, S. (2015). Assessment of the suitability of fog computing in the context of internet of things. *IEEE Transactions on Cloud Computing*, 6(1):46–59.

[119] Satyanarayanan, M. (2010). Mobile computing: the next decade. In *Proceedings of the 1st ACM workshop on mobile cloud computing & services: social networks and beyond*, page 5.

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

[121] Sequence Plugin (2019). Sequence diagram. [online] http://vanco.github.io/SequencePlugin/, last accessed on 03-11-2019.

[122] Shi, C., Ammar, M. H., Zegura, E. W., and Naik, M. (2012a). Computing in cirrus clouds: the challenge of intermittent connectivity. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 23–28.

[123] Shi, C., Habak, K., Pandurangan, P., Ammar, M., Naik, M., and Zegura, E. (2014). Cosmos: computation offloading as a service for mobile devices. In *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pages 287–296.

[124] Shi, C., Lakafosis, V., Ammar, M. H., and Zegura, E. W. (2012b). Serendipity: Enabling remote computing among intermittently connected mobile devices. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, pages 145–154.

[125] Shi, Y., Chen, S., and Xu, X. (2017). Maga: A mobility-aware computation offloading decision for distributed mobile cloud computing. *IEEE Internet of Things Journal*, 5(1):164–174.

[126] Shih, C., Wang, Y., and Chang, N. (2015). Multi-tier elastic computation framework for mobile cloud computing. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 223–232.

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

[128] Shiraz, M., Ahmed, E., Gani, A., and Han, Q. (2014). Investigation on runtime partitioning of elastic mobile applications for mobile cloud computing. *The Journal of Supercomputing*, 67(1):84–103.

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

[130] Silva, F. A., Zaicaner, G., Quesado, E., Dornelas, M., Silva, B., and Maciel, P. (2016). Benchmark applications used in mobile cloud computing research: a systematic mapping study. *The Journal of Supercomputing*, 72(4):1431–1452.

[131] Sinha, K. and Kulkarni, M. (2011). Techniques for fine-grained, multi-site computation offloading. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 184–194.

[132] smartphone-users-still-want-longer-battery-life (2018). Smartphone users still want long-lasting batteries more than shatterproof screens. [online] https://today.yougov.com/topics/technology/articles-reports/2018/02/20/smartphone-users-still-want-longer-battery-life, last accessed on 04-09-2019.

[133] Soyata, T., Muraleedharan, R., Funai, C., Kwon, M., and Heinzelman, W. (2012). Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *2012 IEEE Symposium on Computers and Communications (ISCC)*, pages 000059–000066.

[134] Strobel, V. (2018). Pold87/academic-keyword-occurrence: First release. *Zenodo*.

[135] Sulaiman, D. and Barker, A. (2016). Task offloading engine for heterogeneous mobile clouds. In *Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services*, pages 147–148.

[136] Sulaiman, D. and Barker, A. (2017). Mamoc: Multisite adaptive offloading framework for mobile cloud applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 17–24.

[137] Sulaiman, D. and Barker, A. (2019). Mamoc-android: Multisite adaptive computation offloading for android applications. In *2019 7th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 68–75.

[138] Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., and Sabella, D. (2017). On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681.

[139] Tarkoma, S. (2012). *Publish/subscribe systems: design and principles*. John Wiley & Sons.

[140] Terefe, M. B., Lee, H., Heo, N., Fox, G. C., and Oh, S. (2016). Energy-efficient multisite offloading policy using markov decision process for mobile cloud computing. *Pervasive and Mobile Computing*, 27:75–89.

[141] The Mobile Economy (2018). The mobile economy 2018. [online] http://bit.ly/mobiledesktop2018, last accessed on 12-06-2019.

[142] Toyama, M., Kurumatani, S., Heo, J., Terada, K., and Chen, E. Y. (2011). Android as a server platform. In *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 1181–1185.

[143] u. R. Khan, A., Othman, M., Madani, S. A., and Khan, S. U. (2014). A survey of mobile cloud computing application models. *IEEE Communications Surveys Tutorials*, 16(1):393–413.

[144] Vakali, A. and Pallis, G. (2003). Content delivery networks: Status and trends. *IEEE Internet Computing*, 7(6):68–74.

[145] Varghese, B., Subba, L. T., Thai, L., and Barker, A. (2016). Container-based cloud virtual machine benchmarking. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 192–201.

[146] Velasquez, M. and Hester, P. T. (2013). An analysis of multi-criteria decision making methods.

[147] Verbelen, T., Simoens, P., De Turck, F., and Dhoedt, B. (2014). Adaptive deployment and configuration for mobile augmented reality in the cloudlet. *Journal of Network and Computer Applications*, 41:206–216.

[148] Verbelen, T., Stevens, T., De Turck, F., and Dhoedt, B. (2013). Graph partitioning algorithms for optimizing software deployment in mobile cloud computing. *Future Generation Computer Systems*, 29(2):451–459.

[149] Voigt, P. and Von dem Bussche, A. (2017). The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*.

[150] Wang, Y., Chen, R., and Wang, D.-C. (2015). A survey of mobile cloud computing applications: Perspectives and challenges. *Wireless Personal Communications*, 80(4):1607–1623.

[151] Wi-Fi Alliance (2016). Wi-fi peer-to-peer (p2p) technical specification, version 1.7. [online] https://www.wi-fi.org/file/wi-fi-peer-to-peer-p2p-technical-specification-v17, last accessed on 23-10-2019.

[152] Wu, H. (2018). Multi-objective decision-making for mobile cloud offloading: A survey. *IEEE Access*, 6:3962–3976.

[153] Wu, H. and Huang, D. (2014). Modeling multi-factor multi-site risk-based offloading for mobile cloud computing. In *10th International Conference on Network and Service Management (CNSM) and Workshop*, pages 230–235.

[154] Wu, S., Mei, C., Jin, H., and Wang, D. (2018). Android unikernel: Gearing mobile code offloading towards edge computing. *Future Generation Computer Systems*.

[155] Wu, S., Niu, C., Rao, J., Jin, H., and Dai, X. (2017). Container-based cloud platform for mobile computation offloading. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*, pages 123–132.

[156] Xia, F., Ding, F., Li, J., Kong, X., Yang, L. T., and Ma, J. (2014a). Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers*, 16(1):95–111.

[157] Xia, Q., Liang, W., Xu, Z., and Zhou, B. (2014b). 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*, pages 109–116. IEEE Computer Society.

[158] Yangui, S., Ravindran, P., Bibani, O., Glitho, R. H., Ben Hadj-Alouane, N., Morrow, M. J., and Polakos, P. A. (2016). A platform as-a-service for hybrid cloud/fog environments. In *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–7.

[159] Yaqoob, I., Ahmed, E., Gani, A., Mokhtar, S., Imran, M., and Guizani, S. (2016). Mobile ad hoc cloud: A survey. *Wireless Communications and Mobile Computing*, 16(16):2572–2589. WCM-16-0169.R1.

[160] Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L. (2010). Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES/ISSS '10, pages 105–114.

[161] Zhang, W.-L., Guo, B., Shen, Y., Li, D.-G., and Li, J.-K. (2018). An energy-efficient algorithm for multi-site application partitioning in mcc. *Sustainable Computing: Informatics and Systems*, 18:45–53.

[162] Zhang, X., Yang, Z., Sun, W., Liu, Y., Tang, S., Xing, K., and Mao, X. (2015). Incentives for mobile crowd sensing: A survey. *IEEE Communications Surveys & Tutorials*, 18(1):54–67.

[163] Zhou, B. and Buyya, R. (2018). Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions. *ACM Computing Surveys (CSUR)*, 51(1):13.

[164] Zhou, B., Dastjerdi, A. V., Calheiros, R. N., Srirama, S. N., and Buyya, R. (2015). mcloud: A context-aware offloading framework for heterogeneous mobile cloud. *IEEE Transactions on Services Computing*, 10(5):797–810.

[165] Zoraghi, N., Amiri, M., Talebi, G., and Zowghi, M. (2013). A fuzzy mcdm model with objective and subjective weights for evaluating service quality in hotel industries. *Journal of Industrial Engineering International*, 9(1):38.

Appendix A

Code Transformation Examples

The code transformer model was described in Section 5.7.2.2. This appendix provides examples of the offloadable tasks in their transformations from the Android application code to the server side Java code.

A.1 Transforming Classes

MAMoC client sends over an array of method parameters along with the name of the class to the offloading site. The types are dynamically invoked to be called in the main method of the generated class. As an example, let us demonstrate a text searching task using the Knuth-Morris-Pratt algorithm [72] which is defined in *KMP* class. Listing A.1 shows the original Android code.

```
package uk.ac.standrews.cs.mamoc.SearchText;

@Offloadable(resourceDependent = true, parallelizable = true)
public class KMP {
    String content, pattern;

    public KMP(String content, String pattern) {
        this.content = content;
        this.pattern = pattern;
    }

    public int run() {
        ....
    }
```

Listing A.1 KMP on Android

When the task is executed on the Android application, it is first checked whether the remote procedure is already registered by fetching the list of registered

procedures in the server by calling *wamp.registration.list*. If the procedure exists, we only need to send the parameters and resource names (if any). Otherwise, the following event will be published which the server is subscribed to:

publish("uk.ac.standrews.cs.mamoc.offloading", "Android", "uk.ac.standrews.cs.mamoc.SearchText.KMP", Decompiled KMP SourceCode, "large.txt", searchKeyword)

After the server receives it, the Java code in Listing A.2 will be generated according to Algorithm 5.1 described earlier.

```
import java.nio.file.Files;
import java.nio.file.Paths;
import java.io.IOException;

public class KMP {
    public static String readResourceContent(String filePath){
        try {
            return new String(Files.readAllBytes(Paths.get(filePath)
                ));
        } catch (IOException e) {
            ....
        }
    }

    public static void main(String[] args){
        System.out.print(new KMP(readResourceContent(args[0]), args
            [1]).run());
    }

    String content, pattern;

    public KMP(String content, String pattern) {
        ....
    }
}
```

Listing A.2 KMP on server side

The generated Java code is then compiled using *javac KMP* and executed with *java KMP large hello*. The two passed arguments indicate the name of the text file (large.txt) and the search keyword ("hello") to be found in the text file. The result of execution and duration in milliseconds are then published to the device which have subscribed to *uk.ac.standrews.cs.mamoc.offloadingresult.KMP* topic.

Listing A.3 shows an Android code for another task example for solving N-Queens problem. Since this task does not have a resource dependency, the server

generated code is more straightforward than the previous example, as shown in Listing A.4.

```
package uk.ac.standrews.cs.mamoc.NQueens;

@Offloadable
public class Queens {
    int n;

    public Queens(int N){
        this.n = N;
    }

    public void run() {
        ...
    }
```

Listing A.3 NQueens on Android

```
public class Queens {
    public static void main(String[] args){
        new Queens(Integer.parseInt(args[0])).run();
    }

    int n;

    public Queens(int N) {
        ....
    }
```

Listing A.4 NQueens on server

A.2 Transforming Methods

MAMoC Server accepts task offloading requests in both class and method levels. Therefore, the code transformer should be able to identify the type of the task before transforming it. Listing A.5 shows how the type of the task is identified by checking whether the derived code starts with the keyword *package* that indicates it is a class-level offloading.

```
if code.strip().split(' ', 1)[0] == "package":
    code, self.class_name = Transformer(code, resourcename, params).
        start()
else:
    code, self.class_name = Transformer(code, resourcename, params).
        start(type="method")
```

Listing A.5 Identifying class and method level offloading requests

In order to execute the method correctly, it is wrapped in a test runner class with a *main()* method so that it can be compiled and executed by the execution controller component as demonstrated in both Listing A.6 and Listing A.7 .

```
@Offloadable
public void testMethod(){
    String test = "Testing method offloading";
    System.out.print(test);
}
```

Listing A.6 Method example passed to the code transformer

```
public class TestMethodRunner{

    public void testMethod(){
        String test = "Testing method offloading";
        System.out.print(test);
    }

    public static void main(String[] args){
        new TestMethodRunner().testMethod();
    }
}
```

Listing A.7 Method code transformation on the MAMoC server

The complete source code of MAMoC and a short documentation for setting up the different components in the framework is publicly available online at *https://github.com/dawand/MAMoC-Android*. The repository includes further examples including N-Queens and Fibonacci tasks. The server component which can also be pulled from Docker hub is also available at *https://github.com/dawand/MAMoC-Server*.

Appendix B

The Group Decision Making Results of Each Decision Maker

This appendix presents the steps of fuzzification and site rankings using AHP and fuzzy TOPSIS methods in the GDM scenario that were presented in Chapter 4 and evaluated in Chapter 6. The evaluation of the GDM presented in Section 6.4 aggregated the decision matrices from all the DMs into the group judgment matrix $A^{(G)}$ that were used for evaluating the criteria and then ranking the alternatives.

This appendix utilises the same set of offloading sites presented in Table 6.7 for running the MCDM GDM program for each decision maker individually and presents the individual DM results.

	Bandwidth	**Speed**	**Availability**	**Security**	**Price**
Weights	0.4073	0.3886	0.1084	0.0573	0.0385
Mobile-1	(0.305, 0.389, 0.389)	(0.0, 0.097, 0.194)	(0.097, 0.194, 0.291)	(0.194, 0.291, 0.389)	(0.0, 0.0, 0.097)
Mobile-2	(0.305, 0.389, 0.389)	(0.0, 0.097, 0.194)	(0.0, 0.097, 0.194)	(0.194, 0.291, 0.389)	(0.0, 0.0, 0.097)
Mobile-3	(0.305, 0.389, 0.389)	(0.194, 0.291, 0.389)	(0.0, 0.097, 0.194)	(0.194, 0.291, 0.389)	(0.0, 0.0, 0.097)
Edge-1	(0.204, 0.291, 0.389)	(0.097, 0.194, 0.291)	(0.194, 0.291, 0.389)	(0.194, 0.291, 0.389)	(0.0, 0.097, 0.194)
Edge-2	(0.204, 0.291, 0.389)	(0.194, 0.291, 0.389)	(0.194, 0.291, 0.389)	(0.194, 0.291, 0.389)	(0.097, 0.194, 0.291)
Edge-3	(0.204, 0.291, 0.389)	(0.291, 0.389, 0.389)	(0.194, 0.291, 0.389)	(0.194, 0.291, 0.389)	(0.194, 0.291, 0.389)
Public-1	(0.204, 0.291, 0.389)	(0.0, 0.097, 0.194)	(0.291, 0.389, 0.389)	(0.0, 0.097, 0.194)	(0.097, 0.194, 0.291)
Public-2	(0.102, 0.194, 0.291)	(0.194, 0.291, 0.389)	(0.291, 0.389, 0.389)	(0.0, 0.097, 0.194)	(0.194, 0.291, 0.389)
Public-3	(0.0, 0.097, 0.1943)	(0.291, 0.389, 0.389)	(0.291, 0.389, 0.389)	(0.0, 0.097, 0.194)	(0.291, 0.389, 0.389)

Table B.1 Weighted fuzzy evaluation of offloading sites according to DM1: MCDM-GDM-DM1

Offloading site	D_i^+	D_i^-	C_i^*
Mobile-3	2.7284	1.6862	0.3820
Edge-2	2.6386	1.5978	0.3772
Edge-1	2.6723	1.6048	0.3752
Edge-3	2.6651	1.5753	0.3715
Mobile-1	2.8251	1.5964	0.3611
Mobile-2	2.9219	1.5334	0.3442
Public-1	2.9624	1.3664	0.3156
Public-2	2.9567	1.3310	0.3104
Public-3	3.0657	1.2776	0.2942

Table B.2 Final ranking of the offloading sites according to DM1: MCDM-GDM-DM1

	Bandwidth	**Speed**	**Availability**	**Security**	**Price**
Weights	0.5888	0.2219	0.1178	0.0357	0.0357
Mobile-1	(0.442, 0.222, 0.222)	(0.0, 0.055, 0.111)	(0.055, 0.111, 0.166)	(0.111, 0.166, 0.222)	(0.0, 0.0, 0.055)
Mobile-2	(0.442, 0.222, 0.222)	(0.0, 0.055, 0.111)	(0.0, 0.055, 0.111)	(0.111, 0.166, 0.222)	(0.0, 0.0, 0.055)
Mobile-3	(0.442, 0.222, 0.222)	(0.111, 0.166, 0.222)	(0.0, 0.055, 0.111)	(0.111, 0.166, 0.222)	(0.0, 0.0, 0.055)
Edge-1	(0.294, 0.166, 0.222)	(0.055, 0.111, 0.166)	(0.111, 0.166, 0.222)	(0.111, 0.166, 0.222)	(0.0, 0.055, 0.111)
Edge-2	(0.294, 0.166, 0.222)	(0.111, 0.166, 0.222)	(0.111, 0.166, 0.222)	(0.111, 0.166, 0.222)	(0.055, 0.111, 0.166)
Edge-3	(0.294, 0.166, 0.222)	(0.166, 0.222, 0.222)	(0.111, 0.166, 0.222)	(0.111, 0.166, 0.222)	(0.111, 0.166, 0.222)
Public-1	(0.294, 0.166, 0.222)	(0.0, 0.055, 0.111)	(0.166, 0.222, 0.222)	(0.0, 0.055, 0.111)	(0.055, 0.111, 0.166)
Public-2	(0.147, 0.111, 0.166)	(0.111, 0.166, 0.222)	(0.166, 0.222, 0.222)	(0.0, 0.055, 0.111)	(0.111, 0.166, 0.222)
Public-3	(0.0, 0.055, 0.111)	(0.166, 0.222, 0.222)	(0.166, 0.222, 0.222)	(0.0, 0.055, 0.111)	(0.166, 0.222, 0.222)

Table B.3 Weighted fuzzy evaluation of offloading sites according to DM2: MCDM-GDM-DM2

Offloading site	D_i^+	D_i^-	C_i^*
Mobile-3	3.1465	1.5100	0.3243
Mobile-1	3.2016	1.4798	0.3161
Mobile-2	3.2569	1.4798	0.3124
Edge-1	3.1145	1.3806	0.3073
Edge-2	3.0593	1.3557	0.3071
Edge-3	3.0540	1.3448	0.3057
Public-1	3.2445	1.3396	0.2922
Public-2	3.2404	1.2356	0.2760
Public-3	3.3293	1.2139	0.2672

Table B.4 Final ranking of the offloading sites according to DM2: MCDM-GDM-DM2

	Bandwidth	**Speed**	**Availability**	**Security**	**Price**
Weights	0.1082	0.5213	0.3030	0.0337	0.0337
Mobile-1	(0.081, 0.521, 0.521)	(0.0, 0.130, 0.261)	(0.130, 0.261, 0.391)	(0.261, 0.391, 0.521)	(0.0, 0.0, 0.130)
Mobile-2	(0.081, 0.521, 0.521)	(0.0, 0.130, 0.261)	(0.0, 0.130, 0.261)	(0.261, 0.391, 0.521)	(0.0, 0.0, 0.130)
Mobile-3	(0.081, 0.521, 0.521)	(0.261, 0.391, 0.521)	(0.0, 0.130, 0.261)	(0.261, 0.391, 0.521)	(0.0, 0.0, 0.130)
Edge-1	(0.054, 0.391, 0.521)	(0.130, 0.261, 0.391)	(0.261, 0.391, 0.521)	(0.261, 0.391, 0.521)	(0.0, 0.130, 0.261)
Edge-2	(0.054, 0.391, 0.521)	(0.261, 0.391, 0.521)	(0.261, 0.391, 0.521)	(0.261, 0.391, 0.521)	(0.130, 0.261, 0.391)
Edge-3	(0.054, 0.391, 0.521)	(0.391, 0.521, 0.521)	(0.261, 0.391, 0.521)	(0.261, 0.391, 0.521)	(0.261, 0.391, 0.521)
Public-1	(0.054, 0.391, 0.521)	(0.0, 0.130, 0.261)	(0.391, 0.521, 0.521)	(0.0, 0.130, 0.261)	(0.130, 0.261, 0.391)
Public-2	(0.027, 0.261, 0.391)	(0.261, 0.391, 0.521)	(0.391, 0.521, 0.521)	(0.0, 0.130, 0.261)	(0.261, 0.391, 0.521)
Public-3	(0.0, 0.130, 0.261)	(0.391, 0.521, 0.521)	(0.391, 0.521, 0.521)	(0.0, 0.130, 0.261)	(0.391, 0.521, 0.521)

Table B.5 Weighted fuzzy evaluation of offloading sites according to DM3: MCDM-GDM-DM3

Offloading site	D_i^+	D_i^-	C_i^*
Edge-2	2.3751	1.8750	0.4412
Edge-1	2.3751	1.8786	0.4397
Mobile-3	2.4650	1.9238	0.4383
Edge-3	2.4148	1.8396	0.4324
Mobile-1	2.5947	1.7974	0.4092
Mobile-2	2.7246	1.6860	0.3823
Public-2	2.7661	1.4971	0.3512
Public-1	2.8076	1.4937	0.3473
Public-3	2.8695	1.4085	0.3292

Table B.6 Final ranking of the offloading sites according to DM3: MCDM-GDM-DM3

	Bandwidth	Speed	Availability	Security	Price
Weights	0.0901	0.1185	0.0752	0.6767	0.0395
Mobile-1	(0.068, 0.118, 0.118)	(0.0, 0.030, 0.059)	(0.030, 0.059, 0.089)	(0.059, 0.089, 0.118)	(0.0, 0.0, 0.030)
Mobile-2	(0.068, 0.118, 0.118)	(0.0, 0.030, 0.059)	(0.0, 0.030, 0.059)	(0.059, 0.089, 0.118)	(0.0, 0.0, 0.030)
Mobile-3	(0.068, 0.118, 0.118)	(0.059, 0.089, 0.118)	(0.0, 0.030, 0.059)	(0.059, 0.089, 0.118)	(0.0, 0.0, 0.030)
Edge-1	(0.045, 0.089, 0.118)	(0.030, 0.059, 0.089)	(0.059, 0.089, 0.118)	(0.059, 0.089, 0.118)	(0.0, 0.030, 0.059)
Edge-2	(0.045, 0.089, 0.118)	(0.059, 0.089, 0.118)	(0.059, 0.089, 0.118)	(0.059, 0.089, 0.118)	(0.030, 0.059, 0.089)
Edge-3	(0.045, 0.089, 0.118)	(0.089, 0.118, 0.118)	(0.059, 0.089, 0.118)	(0.059, 0.089, 0.118)	(0.059, 0.089, 0.118)
Public-1	(0.045, 0.089, 0.118)	(0.0, 0.030, 0.059)	(0.089, 0.118, 0.118)	(0.0, 0.030, 0.059)	(0.030, 0.059, 0.089)
Public-2	(0.023, 0.059, 0.089)	(0.059, 0.089, 0.118)	(0.089, 0.118, 0.118)	(0.0, 0.030, 0.059)	(0.059, 0.089, 0.118)
Public-3	(0.0, 0.030, 0.059)	(0.089, 0.118, 0.118)	(0.089, 0.118, 0.118)	(0.0, 0.030, 0.059)	(0.089, 0.118, 0.118)

Table B.7 Weighted fuzzy evaluation of offloading sites according to DM4: MCDM-GDM-DM4

Offloading site	D_i^+	D_i^-	C_i^*
Mobile-3	3.5073	1.3301	0.2750
Mobile-1	3.5367	1.3329	0.2737
Mobile-2	3.5661	1.3437	0.2737
Edge-1	3.4790	1.2839	0.2696
Public-1	3.5376	1.2955	0.2680
Edge-2	3.4387	1.2517	0.2669
Edge-3	3.4171	1.2388	0.2661
Public-3	3.5286	1.2663	0.2641
Public-2	3.5034	1.2571	0.2641

Table B.8 Final ranking of the offloading sites according to DM4: MCDM-GDM-DM4

	Bandwidth	**Speed**	**Availability**	**Security**	**Price**
Weights	0.1126	0.1126	0.0615	0.0392	0.6742
Mobile-1	(0.084, 0.113, 0.113)	(0.0, 0.028, 0.056)	(0.028, 0.056, 0.084)	(0.056, 0.084, 0.113)	(0.0, 0.0, 0.028)
Mobile-2	(0.084, 0.113, 0.113)	(0.0, 0.028, 0.056)	(0.0, 0.028, 0.056)	(0.056, 0.084, 0.113)	(0.0, 0.0, 0.028)
Mobile-3	(0.084, 0.113, 0.113)	(0.056, 0.084, 0.113)	(0.0, 0.028, 0.056)	(0.056, 0.084, 0.113)	(0.0, 0.0, 0.028)
Edge-1	(0.056, 0.084, 0.113)	(0.028, 0.056, 0.084)	(0.056, 0.084, 0.113)	(0.056, 0.084, 0.113)	(0.0, 0.028, 0.056)
Edge-2	(0.056, 0.084, 0.113)	(0.056, 0.084, 0.113)	(0.056, 0.084, 0.113)	(0.056, 0.084, 0.113)	(0.028, 0.056, 0.084)
Edge-3	(0.056, 0.084, 0.113)	(0.084, 0.113, 0.113)	(0.056, 0.084, 0.113)	(0.056, 0.084, 0.113)	(0.056, 0.084, 0.113)
Public-1	(0.056, 0.084, 0.113)	(0.0, 0.028, 0.056)	(0.084, 0.113, 0.113)	(0.0, 0.028, 0.056)	(0.028, 0.056, 0.084)
Public-2	(0.028, 0.056, 0.084)	(0.056, 0.084, 0.113)	(0.084, 0.113, 0.113)	(0.0, 0.028, 0.056)	(0.056, 0.084, 0.113)
Public-3	(0.0, 0.028, 0.056)	(0.084, 0.113, 0.113)	(0.084, 0.113, 0.113)	(0.0, 0.028, 0.056)	(0.084, 0.113, 0.113)

Table B.9 Weighted fuzzy evaluation of offloading sites according to DM5: MCDM-GDM-DM5

Offloading site	D_i^+	D_i^-	C_i^*
Mobile-3	3.5184	1.3380	0.2755
Mobile-1	3.5464	1.3416	0.2745
Mobile-2	3.5743	1.3524	0.2745
Edge-1	3.4933	1.2921	0.2700
Public-1	3.5485	1.3042	0.2688
Edge-2	3.4546	1.2605	0.2673
Edge-3	3.4330	1.2475	0.2665
Public-2	3.5169	1.2655	0.2646
Public-3	3.5419	1.2733	0.2644

Table B.10 Final ranking of the offloading sites according to DM5: MCDM-GDM-DM5

Acronyms

AHP Analytic Hierarchy Process.

AP Access Point.

APK Android PacKage file.

D2D Device-to-Device.

DEX Dalvik Executable format.

DM Decision Maker.

DVM Dalvik Virtual Machine.

GC Group Client.

GDM Group Decision Making.

GO Group Owner.

JVM Java Virtual Machine.

LAN Local Area Network.

MAC Mobile Ad-hoc Cloud. *Glossary:*Mobile Ad-hoc Cloud.

MCC Mobile Cloud Computing. *Glossary:*Mobile Cloud Computing.

MCDM Multi-Criteria Decision Methods. *Glossary:*Multi-Criteria Decision Methods.

MCO Mobile Computation Offloading. *Glossary:*Mobile Computation Offloading.

MEC Mobile Edge Computing.

MFC Mobile Fog Computing.

P2P Peer-to-peer.

Pub/Sub Publish/Subscribe messaging pattern.

RPC Remote Procedure Call.

TOPSIS Technique for Order of Preference by Similarity to Ideal Solution.

VM Virtual Machine.

WAMP Web Application Messaging Protocol.

WAN Wide Area Network.

Glossary

Cloudlet A powerful nearby fixed surrogate to empower the low-powered mobile devices. Also see: Edge Node.

Edge Node The edge server or cloudlet which currently participates in the MAMoC framework as a service provider for Self Nodes.

Host Mobile Device The mobile client which initiaties the offloading request. Also see: Self Node.

MAMoC Client The mobile client library to support Android applications to be MAMoC-enabled for task offloading.

MAMoC Server The server runtime environment deployed to the service providers.

Mobile Ad-hoc Cloud An ad-hoc formation of multiple nearby mobile devices to support local offloading for Self Nodes.

Mobile Cloud Computing An amalgamation of mobile computing and cloud computing for leveraging mobile devices with the compute powers of the vast cloud resouces.

Mobile Computation Offloading The process of moving parts of a mobile application to an external entity for remote execution.

Mobile Node The nearby mobile device which currently participates in the MAMoC framework as a service provider for Self Nodes.

Multi-Criteria Decision Methods A set of decision making techniques that use multiple criteria for finding optimal selection and ranking of the altrnatives.

Nearby Mobile Device The local mobile devices which can form Mobile Ad-hoc Cloud. Also see: Mobile Node.

Public Node The remote cloud instance which currently participates in the MAMoC framework as a service provider for Self Nodes.

Remote Cloud The datacenter regional servers such as AWS, GCP, and Azure. Also see: Public Node.

Self Node The MAMoC-enabled mobile client which can use the MAMoC offloading decision engine.

Task The offloadable class or method that is MAMoC-enabled according to the task specifications discussed in Subsection 5.2.1.