An Energy-Efficient Method for Hybrid Mobile Cloud Computing

Aamir AKBAR

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Aston Lab for Intelligent Collectives Engineering (ALICE)
Department of Computer Science

ASTON UNIVERSITY

JANUARY, 2020

ASTON UNIVERSITY

Department of Computer Science

An Energy-Efficient Method for Hybrid Mobile Cloud Computing

Aamir AKBAR

Doctor of Philosophy, 2020

Abstract

Many benefits of cloud computing are now well established, as both enterprise and mobile IT have been transformed by its pervasiveness. Backed by the virtually unbounded resources of cloud computing, battery-powered mobile computing systems can meet the demands of even the most resource-intensive applications. However, many existing hybrid mobile-cloud (HMC) applications are inefficient in terms of optimising trade-offs between simultaneously conflicting objectives, such as minimising both battery power consumption and network usage. To tackle this problem, we propose a novel method that can be used not only to instrument HMC applications but also to search for its efficient configurations representing compromise solutions between the objectives. The method is based on a general purpose HMC framework, which provides scalability, and make runtime decisions that are based on: 1) changing of the environment (i.e. WiFi signal level variation), and 2) itself in a changing environment (i.e. actual observed packet loss in the network). Our experimental evaluation considers two Android-based applications for smartphones, and a Python-based foraging task performed by a battery-powered and Raspberry Pi controlled Thymio robot. Analysis of our results shows that our method can be used for small to medium size HMC applications to achieve energy-efficient computing systems. Furthermore, HMC applications can achieve better optimisation in a changing environment (i.e. signal level variation) than using static offloading or running the applications only on a mobile device. However, a self-adaptive decision would fall behind when the change in the environment happens within the system (i.e. network congestion). In such a case, a self-aware system can perform well, in terms of minimising the two objectives and better performance of applications.

Keywords: Mobile-Cloud Computing, Multi-Objective Optimisation, Cloud Robotics, Self-adaptive Computing Systems, and Self-Aware Computing Systems.

Acknowledgements

All praise due to Almighty God for all his blessings, an important of which was the ability to undertake this PhD. Knowledge is a journey that needs to be taken, and this journey will not be possible without the help and support of others.

At this moment of accomplishment, I would like to thank my supervisor, Dr Peter Lewis. This work would not have been possible without his guidance and involvement, his support and encouragement from the start of the project till date. Under his guidance I successfully overcame many difficulties and learnt a lot. For all these, I sincerely thank him from the bottom of my heart and will be genuinely indebted to him throughout my lifetime. I would also like to express my gratitude to, Dr Elizabeth Wanner, for sharing her insights during the course of this work. In addition, I would also like to thank all the researchers in the ALICE research group with whom I discussed my work and considered their insights and comments.

Nobody has been more important to me in the pursuit of this project than the members of my family. I would like to give my warmest appreciations to my beloved parents, whose love and guidance are with me in whatever I pursue. Who tirelessly supported me in every step of this challenging journey with their immense love, support and sacrifice. They are the ultimate role models. Most importantly, I owe thanks to a very special person in my life, my beloved wife, Aqsa for her love and constant support, for all the late nights and early mornings, and for keeping me sane over the past few months. Nevertheless, I should also thank my aunt, Sarwat Saeed, who provided me all the support and make me feel like at home in the UK. I would also like to thank my beloved sisters, brother, my mother-in-law and grandparents who, despite their physical distance, encouraged me in every step of this journey. I will owe and love all my family members forever.

Finally, this work would not have been possible without the financial support of Abdul Wali Khan University Mardan (AWKUM) in Pakistan, for which I am grateful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, battery-powered mobile devices such as smartphones are used for various tasks. When they were first introduced, many years ago, they were used mainly for telephony purpose. Today's mobile phones can not only be used for telephony, but they are also equipped to be used for a wide range of other applications. A broad category of these applications includes graphics intensive games, GPS navigation and online social networks. With the recent advancement of technology in mobile devices, they are now also equipped with services such as location and context awareness, and the use of sensors (gyro, accelerometer, heart rate, fingerprint and iris scanner). Moreover, mobile phones are also used to create and edit multimedia contents (video and audio) and execute other complex and useful applications. Although enriched with the broad range of services, mobile devices still cannot be exploited to their full potential due to battery life. From a user's point of view, a smartphone can never have enough battery life.

Another group of battery constrained mobile devices are mobile cyber-physical systems (CPS), e.g., operation robots, delivery drones. They combine the physical world with cyber components and have been a key research area for more than ten years [114]. The battery power is crucial in mobile robots when they are deployed for a mission. Moreover, mobile robotic systems have brought significant socio-economic impacts to human lives over the past few decades [116]. For example, robots deployed for rescue missions such as firefighting, natural disasters, hostage situations, and explosions. Mobile CPSs built around a robotic environment have been very successful since they have precision and speed.

However, due to resource-hungry applications executing on mobile devices (i.e. smartphones and robots), they are inherently resource-constrained [113] in computation, energy, and network bandwidth. In particular, the energy supply from the limited battery capacity [98, 131] has been one of the most challenging design issues with mobile devices. When the applications on a mobile device execute, they utilise the device's hardware components to perform. The hardware components (i.e., processor, memory, transceiver, the display of the device) require power to operate, which they get from the device's battery. The more these components are used, the more power is consumed. This leads to short battery life for a mobile device. Therefore, design decisions for mobile applications have to take consideration of the battery power limitation in the mobile device.

In recent years, there has been a growing interest to bind virtual resources to low-power mobile devices [26, 100]. To make mobile devices (i.e. smartphones/tablet, robots) virtually limitless in terms of processing power, energy and storage space, the integration of mobile applications with cloud [20] is often done. This interdisciplinary domain is called Mobile-Cloud Computing (MCC) [72]. Using MCC, parts of mobile application that use high battery power can be offloaded to the cloud. Using the code offloading, the power consumption can be reduced with the cost of using an available network to the mobile device. However, as the Internet is now cheaply available to users and also there are free WIFI offered, in public spaces such as parks, city centres, malls, hotels, restaurants, in transport services and in universities. Therefore, in this work, we relax the constraint of the network usage cost.

In MCC, "cloud" can refer to both virtual and real clouds. Virtual cloud refers to the virtual machine (VM) instances, which are powered by VMWare Workstations or Oracle VM VirtualBox. They can be installed on a computing system such as desktop computer for a lab-based environment to provide services to mobile devices. Real cloud refers to the traditional cloud infrastructure that provides virtually unlimited resources such as IBM cloud [137], Rackspace [10], Amazon Elastic Compute Cloud (EC2) [2], Google App Engine [6]. Real cloud also includes multiple cloud deployment models, such as community, private, public, and hybrid [125].

Mobile cloud computing (MCC) is a distributed and augmented program execution model, upon which hybrid mobile-cloud (HMC) applications are developed [53]. Such applications have the advantage of using the vast majority of services provided by both mobile and cloud computing. Whereas the fundamental goal of cloud computing is to provide IT support to institutions in a cost-effective way, MCC concentrates on overcoming mobile devices constraints and enhancing mobile users' experience. Moreover, the HMC applications are becoming ever more frequently relied upon. For example, to find a route between two locations a navigation application installed on a mobile device can use the stored maps, but to find the real-time traffic on that route the application would need the cloud service. Some of the advantages of MCC are discussed as follow [125].

1. MCC utilises the computational power of the cloud, therefore, enhances the performance of the mobile device's applications.
2. MCC can extend the battery life of mobile devices, as the applications are partly executed on the cloud. Therefore, reducing the computational overhead on the devices.
3. MCC can enable the resource-constrained mobile devices to execute resource-intensive applications.

In MCC, the battery-power consumption is reduced by using computation offloading [87]. The primary aim of offloading is the migration of computationally-intensive tasks from a mobile device to execute on the cloud. Traditionally, code-offloading was not employed in the mobile applications development models since they were intended to be executed on the device only. In order to offload the computationally-intensive tasks, a mobile device can connect to the cloud via a wireless network (i.e., WIFI, 3G/4G). From a connection point of view: the decision to offload to the cloud can be affected by the following two factors.

1. Network delay (latency) due to a long distance in terms of network topology between a mobile device and the conventional cloud.
2. A high network usage cost, e.g., when an application offloads to the cloud too often or a large amount of data is transferred between a mobile device and the cloud.

In order to overcome the delay caused due to long distance, Mobile Edge Computing (MEC) has recently emerged [60] and promises to provide low latency due to be in proximity to mobile devices. An Edge device is local to the mobile devices (in terms of network topology) where data is generated and collected. MEC primarily aims to overcome the problem of latency, which is one of the shortcomings in MCC. Furthermore, due to highly efficient network operation and service delivery, MEC offers an improved user experience. Resource-hungry applications/services such as augmented reality, intelligent video acceleration, and connected cars can benefit from MEC. However, MEC has limited computing capabilities compared to the conventional cloud computing [94]. Alternatively, a virtualised platform called Mobile Fog Computing [28] has recently emerged that provides computing services between mobile devices and the cloud data centres.

In case of whichever underlying platform being used, the code-offloading of computationally-intensive tasks can also be affected by a high network usage. Furthermore, the transmitter chip (WIFI, 3G/4G etc.) also uses the battery power when used for code offloading. As a result, relying too much on code-offloading will also use more battery power. Attaining minimum battery power consumption and network usage, while not affecting the overall performance of mobile applications, is considered one of the challenging areas in mobile-cloud computing [13]. Therefore, we consider hybrid mobile-cloud applications to have an *efficiency trade-off* between power consumption and network usage. The efficiency trade-off exists because of the following two factors.

- Performing computationally-intensive tasks on mobile devices can be inefficient in terms of battery power consumption.
- Transferring data between a mobile device and the cloud can be inefficient in terms of network usage cost and battery power consumption by the transmitter chip of a mobile device.

Achieving two or more objectives at the same time might not be possible. For example, minimising network usage may prevent the objective of minimising power consumption because the transceiver chip also uses power to send or receive data packets. We consider the effective partitioning of hybrid mobile-cloud applications as a multi-objective optimisation (MOO) problem. In multi-objective optimisation, there are more than one objectives to be optimised simultaneously, and typically the objectives are conflicting with each other. There exists a natural trade-off between the objectives, which creates a set of Pareto-optimal solutions [41], those that are not dominated by any possible other solution in the solution space. Therefore, we consider the following two objectives to optimise.

1. Minimise power consumption: the total power consumption of an application on a mobile device, during one execution. We will measure it in joules (the unit of energy).

2. Minimise network usage: the total amount of data transfer (data sent and received) between a mobile device and the cloud endpoint, during one execution. We will measure it in KBs.

To optimise the two objectives, we proposed a technique [15, 16, 17] to find and apply the optimal configurations of an HMC application. The optimal configurations are considered to be the ones in which the application has minimal battery power consumption and minimal network usage. A *configuration* is a valid mapping of all distinct and offloadable modules of an HMC application to be executed on mobile and the cloud server endpoints. We assume that an application is composed of a set of collaborative code units called modules (i.e., classes or methods in Java and Python). We annotate the computationally-intensive modules that are heavily used at the code level. To automatically convert them into offloadable modules, we use a converter tool. By using a simple HMC application framework, the offloadable modules can be executed both locally on a mobile device and remotely on the cloud server.

The proposed technique is based on the standard TCP/IP sockets to offload data to the cloud. It is up to the developer to implement more secure protocols like *SSL* to protect the user data during communication. Also, developers when using the proposed technique may provide a mechanism for users to avoid offloading of certain modules to the cloud, particularly those modules that contain user's critical data.

In the end, the proposed solution is general purpose and is currently only limited to Android-based and Linux-based applications developed for mobile devices. The HMC framework can be used by developers, service providers and companies to develop energy-efficient applications for the users. For example, a service provider (i.e., Vodafone) can provide their customers with an alternative option using this framework, which would minimise the mobile battery consumption.

1.1 Aims, Contributions to Knowledge and Results

This work aims to achieve energy efficient hybrid mobile-cloud (HMC) applications for Android-based and Linux-based mobile computing systems. The HMC applications execute computationally-intensive modules on the cloud to save battery power, therefore, using network bandwidth. This creates efficiency trade-off between power and network usage. This work aims to optimise the efficiency trade-off. Also, this work aims to provide a scalable solution in terms of achieving energy efficient applications proportion to their size. Finally, the environment in which mobile devices operate can change with time, i.e., network congestion can cause delays to send/receive data from the cloud. This work aims to provide a solution so that applications can adapt to a changing environment.

1.1.1 Contributions

The following paragraphs highlight the main contributions of this work.

Contribution 1: In this work, we propose a novel method that effectively partition mobile applications created for Android-based and Linux-based devices. The aim of the partitioning is to achieve energy-efficient HMC applications using different configurations. For this to achieve, we use a general purpose framework. The framework has two versions: for Android-based applications targetting smartphones, and for Python-based applications targetting mobile robots running a Linux OS distro. Using a configuration, the framework decides at runtime which modules to offload to the cloud. Also, the framework provides APIs to partition mobile applications. Furthermore, we discuss the process of partitioning applications, granularity and configurations of HMC applications in Chapter 3.

Contribution 2: We provide a solution to optimise the trade-off between power consumption and network usage, using multi-objective optimisation (MOO). We have created a workflow, which is based on offline profiling of applications, to find efficient configurations of the applications. Furthermore, we use statistical tests to achieve efficient HMC applications in terms of optimising the trade-off between battery power consumption and network usage. Therefore, the final configurations after the tests are: 1) statistically-significant, and 2) non dominated by other configurations. We achieve this in Chapter 4.

Contribution 3: We provide a scalable solution to achieve energy efficient HMC applications. We have proposed a Two-Step search algorithm, which is based on multi-objective optimisation, to find approximate Pareto-optimal configurations for HMC applications in a feasible amount of time. We discuss this in Chapter 5.

Contribution 4: Our framework for HMC applications is based on self-adaptation and self-awareness, which is to make runtime decisions based on a change in the environment or change within itself in that changing environment. Based on the decisions, the framework switches between the configurations to optimise the efficiency trade-off and avoid latency in the network. We have created a workflow that is based on online profiling of the applications to test this behaviour of the framework. We achieve this in Chapter 6.

1.1.2 Results of Experimental studies

In this work, we carried out experimental studies in order to achieve energy efficient, scalable, self-adaptive and self-aware hybrid mobile-cloud computing systems. We discuss the results as follow.

1. Partitioning of applications was achieved using our MC framework. The application's code were modularised into different levels of granularity. We created configurations to represent these modules, and their endpoints for execution. This has been illustrated in Chapter 3.

2. Our method is based on multi-objective optimisation and code-offloading techniques. Using which the Pareto efficient configurations (in terms of minimising the two objectives) were obtained, which consists of different granularity levels. Furthermore, the results have shown that the reduction in power consumption can be up to 63% in joules, with the cost of using 1.07 MB of the network. This has been achieved in Chapter 4.

3. The Two-Step search algorithm, which we developed, can produce better solutions compared to the current state-of-the-art NSGA-II algorithm [42] for small to medium-scale HMC applications. Small-scale refers to those applications that have less than 8 number of modules. Medium-scale applications have more than 8 and less than 15 number of modules. This is shown in Chapter 5.

4. Based on self-adaptation and self-awareness, a system can achieve minimum power consumption and can also avoid network latency caused by packet loss due to interference, which reduces the network usage. However, the self-adaptive based decisions struggle when the packet loss is due to other factors such as network congestion or link failure casing high packet usage. In such a scenario, the self-aware based decision can achieve minimum power consumption and avoids latency caused by either low signal level or congestion, which minimises the network usage. This is shown in Chapter 6.

1.2 Thesis Structure

This thesis starts by investigating the context of mobile computing systems, cloud computing and mobile-cloud computing. The practices in computation offloading and various approaches towards mobile-cloud applications frameworks are then discussed. These will be covered in Chapter 2: *Background*.

In Chapter 3, *Achieving Energy-Efficient Applications for Mobile Computing Systems*, we use the foundation built in Chapter 2, to describe partitioning of applications and our hybrid mobile-cloud framework. We also introduce Android-based applications and a task performed by a Raspberry Pi controlled robot. We will use these applications throughout the thesis. Chapter 4, 5 and 6 will cover the actual findings. Chapter 4, *Multi-Objective*

Optimisation of Hybrid Mobile-Cloud Applications: will explore the search-based approach to find efficient configurations of HMC applications. Chapter 5, *Scalable Hybrid Mobile-Cloud Computing Systems*, will explore the scalability of the HMC framework. Chapter 6, *Self-Adaptive and Self-Aware Hybrid Mobile-Cloud Computing Systems*, will explore the self-adaptive and self-aware decision mechanism we used to achieve energy-efficient computing systems. Chapter 7 is the *Conclusion* in which we provide an overview of what we have been achieved and possible areas for future work.

1.3 Research Publications

The following publications arose from this work.

1. Peer-reviewed: A. Akbar and P. R. Lewis. **Towards the optimization of power and bandwidth consumption in mobile-cloud hybrid applications**. *In Proceedings of the 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 213–218, May 2017.

2. Peer-reviewed: A. Akbar and P. R. Lewis. **The importance of granularity in multiobjective optimization of mobile cloud hybrid applications**. *Transactions on Emerging Telecommunications Technologies*, pages 221–248, Oct 2018.

3. Peer-reviewed: A. Akbar and P. R. Lewis. **Self-adaptive and self-aware mobile-cloud hybrid robotics**. *In 2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, pages 262–267, Oct 2018.

Chapter 2

Background

In this chapter, we will discuss the background of technologies related to achieving energy-efficient applications for mobile computing systems. In particular, we will discuss methods developed to achieve energy-efficiency in resource-constrained mobile devices. Technologies such as cloud, edge and fog computing provide virtual resources, which can benefit resource-constrained mobile devices. We will discuss the approaches that have been adopted to address energy efficiency such as multi-objective optimisation and computation offloading, and give the reader a broader view of all the related areas and disciplines.

2.1 Cloud Computing

Cloud computing is to provide computing services such as processing, storage capacity, databases, networking and more over the Internet ("the cloud"). The formal definition of cloud computing is stated by the National Institute of Standards and Technology (NIST). It took 15 drafts after which this definition was agreed on [1]. *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*. The NIST definition lists five essential characteristics of cloud computing as follow.

1. Cloud computing should provide *on-demand self-service*. A client can directly change computing capabilities, such as server time and network storage, without requiring human interaction.

[1]https://nvlpubs.nist.gov/ nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

2. Cloud computing workstations should have *broad network access*. By using the standard mechanism, the services and capabilities provided should be accessed over the network.

3. Cloud computing workstations should have a *metering capability*. Involving monitoring and controlling resource usage, and providing transparency for both the provider and client who is utilising the service.

4. Providers of cloud computing services should enable *resource pooling*. Different physical and virtual resources, such as processing, memory, and network bandwidth, are dynamically assigned to serve multiple clients.

5. Cloud computing needs to have *rapid elasticity*. In order to scale rapidly outward and inward commensurate with demand, capabilities of cloud computing can be elastically provisioned and released.

Cloud computing has gained very high popularity by providing high performing, flexible, low cost and on-demand computing services [20, 31]. It has evolved and has also thrived at the same time. Cloud architecture offers three different models for providing its services as follow.

1. **Software as a Service (SaaS) [20]**. To provide the cloud capabilities to clients, so that they can use the provider's applications running on a cloud infrastructure. The clients can access applications using different software such as web browsers or mobile applications.

2. **Platform as a Service (PaaS) [20]**. To provide the cloud capabilities to clients, so that they can deploy onto the cloud infrastructure. The clients can also acquire applications created on the cloud by using programming languages, libraries, services, and tools supported by the provider. The clients are restricted to manage or control the underlying cloud infrastructure. They are only given access to deploy applications or configure settings for the application-hosting environment.

3. **Infrastructure as a Service (IaaS) [20]**. To provide the cloud capabilities to clients, so that they can manage the computing resources such as deploying operating systems and applications. The clients can also manage processing, storage, or networking capabilities, but are restricted to control the underlying cloud infrastructure.

As shown in Figure 2.1, technology giant such as Microsoft [8], Amazon [2], IBM [137] and Google [6] provide cloud-based solutions to their clients in all three layers of the cloud - IaaS, PaaS and SaaS. Moreover, the clients using the cloud-based services have access regardless of whether they are at home or workplace. Besides, to develop new applications on cloud, migrating legacy systems to the cloud or cloud-enabled environments has also been considered [134]. Finally, the design outlines such as Service Oriented Architecture (SOA) [104] are followed, whether developing new cloud-based applications or migrating the legacy system to the cloud.

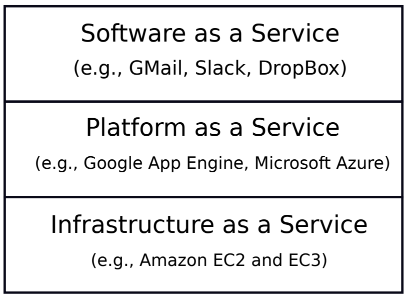

Software as a Service
(e.g., GMail, Slack, DropBox)
Platform as a Service
(e.g., Google App Engine, Microsoft Azure)
Infrastructure as a Service
(e.g., Amazon EC2 and EC3)

FIGURE 2.1: Service-oriented cloud computing architecture.

2.2 Mobile Computing

Mobile computing refers to the use of a computing workstation, commonly known as a mobile device, which has a local storage capacity as well as a wireless network connection based on WiFi (wireless LAN) or cellular (wireless WAN) technology. Mobile computing devices include, but are not limited to, smartphones (i.e. iPhone), tablet computers (i.e. iPad) or battery-powered robots. Based on mobility constraints, Satyanarayanan [113] characterised the mobile devices as follow.

- Mobile devices components (i.e. processor speed, memory size and disk capacity) are resource-poor relative to static components in desktop computers.

- Mobile devices are more vulnerable to loss or damage.
- A mobile device may have to experience a low-bandwidth wireless network with high latency. The chances of low-bandwidth are more likely in remote areas, where there are gaps in coverage.
- Mobile devices rely on limited battery power.

Mobile devices (e.g., smartphones, robots) are frequently used these days. Their popularity has increased mainly because of their support for a wide range of applications/services such as image/video processing, location awareness, context awareness and sensors. An additional advantage of mobile devices is that they can be used in remote areas, where there is no pre-existing infrastructure but it can be deployed during emergency situations [126]. Notably, mobile robots that operate in extreme and high-risk conditions, for example, seal a leak in a nuclear reactor or coordinate search and rescue missions when natural disasters such as earthquakes occur. Unlike humans, mobile robots can be deployed in dangerous sites with little risk.

There are some challenges faced by mobile computing, regardless of the numerous advantages as we have mentioned above. In mobile computing, the same level of performance as in desktop software systems can be challenging to achieve due to limited resources (i.e., battery life, computational power) available to the mobile devices [13]. However, with the advent of technologies such as cloud, edge and fog computing, the applications developed for mobile devices can now access and use their services for resource-intensive tasks [88]. This can minimise the battery power consumption of a mobile device while using the available network.

2.2.1 Mobile Cloud Computing

Mobile clients can interact with and use the cloud services provided by vendors (e.g. Dropbox) using either thin clients (e.g., a web browser) or native mobile applications. The thin client applications such as Google Chrome are developed using standard web development languages (e.g. HTML and JavaScript). The native mobile applications to access the cloud services are developed in mobile platform supported programming languages and API's provided by the cloud service provider. A mobile device that has Internet connectivity (i.e., WiFi, 3G/4G) can access cloud services. As cloud computing has been involved in expanding the IT infrastructure and services for the last few years, the recent advancement of technology in mobile devices (e.g. smartphones, tablets, robots) has also equipped mobile devices in services like; location awareness, context awareness, and the use of camera and sensors (i.e. gyro, accelerometer, fingerprint, iris). To use the features of both cloud computing and mobile computing, the prerequisite was met with the emergence of high-speed broadband and cellular service providers data networks (i.e. LTE, HSPA+, HSPA) to bring mobile and cloud computing together.

Mobile Cloud Computing (MCC) is a distributed and augmented program execution model. The mobile applications developed to use MCC are designed in such a way that the resources of cloud computing are accessed to execute the resource-intensive part of the applications. Therefore, using MCC the resource limitations in mobile devices can be reduced. A formal definition of MCC, stated by Dinh et al. [44], is as follow.

'Mobile cloud computing at its simplest refers to an infrastructure where both the data storage and data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and MC to not just mobile users but a much broader range of mobile subscribers'

From the concept of MCC, the general architecture of MCC can be as shown in Figure 2.2. Mobile devices such as smartphones or robots are connected to the wireless networks via a base station (e.g., base transceiver station or WiFI access point) to access the cloud services using Hybrid Mobile-Cloud (HMC) applications. In general, the services provided by the cloud are known to be promising solutions for HMC applications. Some of the advantages are as follow.

1. **Extending battery life**. Although enriched with the wide range of services, mobile devices still cannot be exploited to their full potential due to battery life. From a user's point of view, a device can never have enough battery life. When the applications on a mobile device execute (mobile computing), they utilise the device's hardware components to perform. The hardware components (i.e., processor, memory, the display of the device) require power to operate, which they get from the device's battery. The more these components are used, the more power

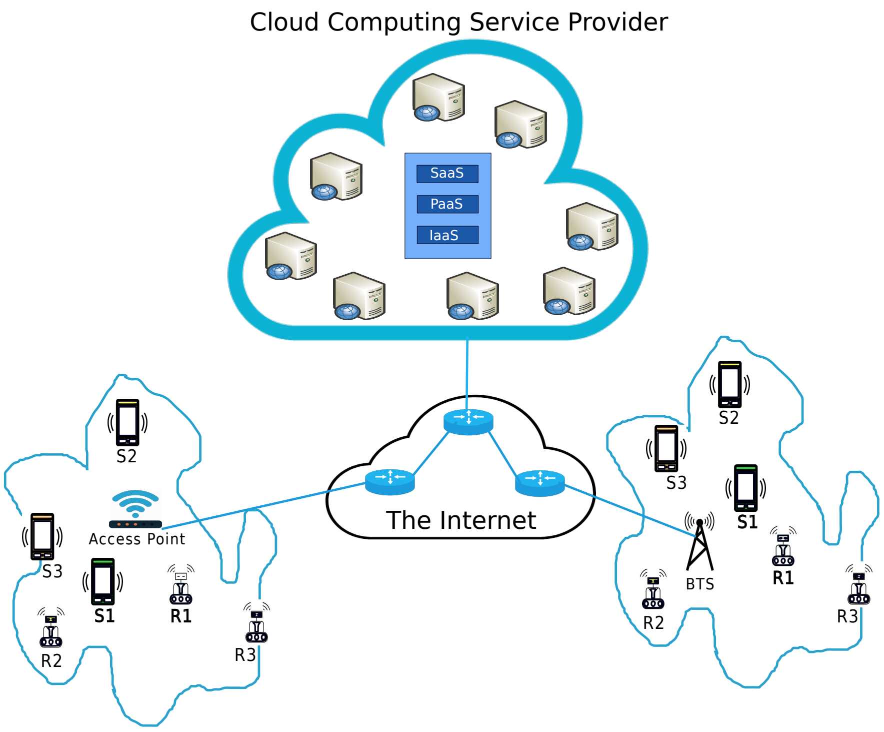


FIGURE 2.2: Mobile Cloud Computing architecture.

is consumed. This leads to short battery life for a mobile device. To minimise the battery power consumption, the computationally-intensive or resource-intensive parts of the HMC applications are offloaded to the cloud for execution. As they execute remotely, the device components are not utilised, and the power is saved. This prolongs battery life.

2. **Improving storage capacity**. Like battery life, the storage capacity is also a constraint for mobile devices. Services like Dropbox [4], Amazon Simple Storage Service [1], Google Photos [7] and Firebase [5] are based on cloud computing that mobile clients can access using MCC to store and share files (i.e. documents, images, videos).

3. **Improving reliability**. Data stored on the cloud can effectively improve the reliability. This is because the data and applications are stored on many computers.

The design of HMC application is based on achieving a particular objective, such as energy efficiency, using storage facilities of cloud, enhancing application performance.

Achieving one objective may affect others. In other words, the objectives to achieve are conflicting and form a trade-off. For example, if the objective of an HMC application is to achieve energy efficiency, then the objective of performance or minimum network usage may be sacrificed. The objective of energy-efficiency and improving performance can be achieved by using computation offloading to execute the resource-intensive part of an application in the cloud [35, 39, 76, 79]. In [15], we proposed a technique that considers two conflicting objectives:

- minimising battery power consumption by HMC applications
- minimising network usage by HMC applications.

We used multi-objective optimisation and code-offloading techniques to find efficient partitions of HMC applications that optimise the trade-off between battery power consumption and network usage. As the WIFI networks are now mostly available freely in many public places such as malls, restaurants, transportations; we relaxed the constraint of network usage cost.

Besides many advantages of MCC, there are some technical challenges that MCC face. They exist since MCC is the integration of two different fields, which are cloud computing and mobile networks. In [44, 77], the challenges faced by MCC are reviewed with available solutions. We highlight the important ones as follow.

1. **Network usage**. It is the amount of data transferred between the mobile device and the cloud server. In MCC, the bandwidth available to mobile devices is always scarce as compared to wired network counterparts. The 4G technology also known as LTE [57] helps to reduce the bandwidth gap between wireless and wired networks. However, the 4G data from the cellular provider may never be free. Therefore, it is important to minimise network usage in MCC.

2. **Service Availability**. It is one of the concerns in MCC. Due to certain factors such as congestions, network failure, wireless interference (out-of-signals), mobile devices may not be able to connect and send data to the cloud.

3. **Latency**. In MCC, latency refers to the round trip time of computation offloading to the cloud and getting back the results. Multiple factors such as offloading data size,

execution delay, network bandwidth and long distance between a mobile device and the cloud can cause latency.

2.2.2 Mobile Edge Computing

Recently a paradigm shift has occurred in mobile computing due to the increasing popularity and vision provided by the Internet of Things (IoT) [23] and 5G [58, 71] cellular communication technologies. From the centralised virtual cloud, mobile computing has shifted towards *Mobile Edge Computing* [60] [30]. In Table 2.1, the comparison between MEC and MCC systems is shown.

TABLE 2.1: Mobile Edge Computing Vs. Mobile Cloud Computing. Source [95]

	Mobile Edge Computing	**Mobile Cloud Computing**
Server hardware	Small-scale data centers with moderate resources [60, 30]	Large-scale data centers (each contains a large number of highly-capable servers) [77]
Server location	Co-located with wireless gateways, WiFi routers, and LTE BSs [60]	Installed at dedicated buildings, with size of several football fields [27]
Deployment	Densely deployed by telecom operators, MEC vendors, enterprises, and home users. Require lightweight configuration and planning [60]	Deployed by IT companies, e.g., Google and Amazon, at a few locations over the world. Require sophisticated configuration and planning
Distance to end users	Small (tens to hundreds of meters) [71]	Large (may across continents) [36]
Backhaul usage	Infrequent use Alleviate congestion [123]	Likely to Creaugsneec-not:gseestion [123]
System management	Hierarchical control (centralized/distributed) [130]	Centralized control [130]
Supportable latency	Less than tens of milliseconds [71]	Larger than 100 milliseconds [39]
Applications	Latency-critical and computation-intensive applications, e.g., AR, automatic driving, and interactive online gaming [60].	Latency-tolerant and computation-intensive applications, e.g., online social networking, and mobile commerce/health/learning [12].

After 5G cellular communication technology is deployed, relying only on cloud computing would not be enough where the data exchange between end mobile devices and remote clouds are stored in centralised locations. Instead, MEC promises to push computation, storage and network control to the edge of the network (e.g. wireless access points, cellular base stations). As the long distance in terms of network topology between a mobile device and the conventional cloud remain a significant drawback for MCC, MEC provides low latency due to be in proximity to mobile devices. In the context of MEC, an edge device (e.g. access point, BTS) is local to the mobile devices (in terms of network topology) where data is generated and collected.

However, MEC has limited computing capabilities compared to the conventional cloud computing [94]. Moreover, security is another critical challenge to the successful deployment of MEC [14]. As the same physical resources are shared among different users, security concerns are raised. It is possible to transfer the data when using computation offloading securely, but the encryption and decryption cause more delay execution of an application, which degrades the application performance.

2.2.3 Mobile Fog Computing

Fog computing [28] is a virtualised platform that provides computing services between mobile devices and the cloud data centres, typically, but not exclusively located at the edge of the network. Fog computing is emerged due to the need for making network edge nodes resource rich. Applications with low latency requirements such as graphics-intensive online gaming, video streaming and augmented reality are beneficiary of Fog computing. Similar to Cloud, Fog provides data, compute, storage, and application services to end-users. Although both cloud and fog computing promise to provide similar capabilities, the latter aims to provide low latency with a wider spread and geographically distributed nodes. While fog computing minimises latency and reduces the amount of data sent to the cloud, it poses security and privacy concerns [120].

2.2.4 Cloud Robotics

Over the past decades, robotics has emerged as a result of its increased applications to numerous real-world problems. This includes, but are not limited to, unmanned search and rescue operations, automated manufacturing, self-driving vehicles, disaster robotics and medical robots. In all these applications, the robots used are limited by their on-board resources (e.g. CPU, memory, storage). To address this problem, researchers have recently proposed cloud-enabled robotics technology. Backed by the virtually unlimited and on-demand resources of cloud computing, cloud robotics integrates the advantages of cloud computing onto robotics.

First coined by James J. Kuffner in 2010 [82], the term '*Cloud Robotics*' refers to a robot or automation system that uses the resources provided by cloud computing for either data storage or computation offloading. For example, a system where all sensing, computation and memory are not integrated into a single stand-alone system.

As discussed in [112], the integration of cloud computing with robotics have several advantages. We highlight them as follow.

1. The computationally-intensive tasks in cloud robotics such as object recognition, computer vision and pattern matching are offloaded to the cloud for execution. Therefore, extending the battery life of mobile robots [128].

2. Cloud-enabled robots have virtually available high storage space to store data. Many applications in robotics, i.e. simultaneous localisation and mapping (SLAM), generate a large amount of sensor data that is difficult to store with the limited on-board storage capacity on most robots [70].

3. Integrating cloud computing to robotics can enable robots to access big data such as global maps for localisation, object models that the robots might need for manipulation tasks as well as open-source algorithms and code [75].

2.3 Mobile Applications Partitioning

When designing a hybrid mobile-cloud application, we are faced with a decision about which module of the application should be executed locally and which one remotely. In order to integrate a stand-alone application created for a mobile computing system with the remote computing service (i.e. the cloud), the source code of the application is partitioned into offloadable modules. This can be achieved by using application partitioning algorithms (APAs) [91]. When the partitioning is done during development time, the code units (i.e. classes/methods) are annotated and using static analysis of the code they are converted into offloadable modules automatically with a converter. On the other hand, when the partitioning is required during execution time, a profiler is used that decides on the fly which modules to execute on the cloud. Either type of partitioning algorithms aims to identify the most computationally-intensive modules for remote processing [61, 87].

One of the important aspects of APAs is the *partitioning granularity* attribute, which indicates the granularity level for partitioning computationally-intensive modules [91]. Gu et al. [62] uses class-level granularity to identify computationally-intensive modules. Cuervo et al. [39] used method-level partitioning of applications. Moreover, a thread-level [35] and object-level [122] partitioning have also been used to offload the computationally-intensive parts of applications to a remote computing server. In [16], we have shown the static partition (at development time) of application code into different levels (class, method and hybrid) of granularity. We highlighted the importance of granularity for efficient partitioning of the applications. We will explain the process of partitioning the code in Chapter 3.

2.4 Code Offloading Techniques

Code offloading (also called computation offloading) is a technique used to transfer the computationally-intensive part of a mobile application to execute on remote locations, typically a resource-rich cloud computing system [61, 87]. In Mobile Cloud Computing (discussed in Section 2.2.1), a key challenge is how to achieve an energy-efficient code offloading [52, 63]. Furthermore, MCC is envisioned to address challenges like extending battery power of mobile devices. With the emergence of fast fibre broadband and high-speed wireless networks (i.e. WiFi, 4G or even 5G), MCC approach to address such challenges.

Code offloading is productive when the battery power consumption of a mobile device is minimum, and counterproductive when the device wastes more energy executing a computational task remotely rather than locally. The experimental study in [111] concluded that code offloading is not always an effective way to save energy. Code offloading might consume more battery power than executing on the mobile device when the size of the code is small.

Among the different techniques used for code offloading, REST, socket-based and RMI-based communications are the most popular and frequently used. For the Android-based mobile applications, since the Java RMI is not supported by *Dalvik*, a new open source and lightweight version is implemented called *LipeRMI* [3, 55]. Moreover, LipeRMI addresses and minimises the latency and communication issues in Java-based RMI. However, the experimental study in [32] resulted that LipeRMI is more costly than REST and socket-based communications.

The decision to use code offloading can be static or dynamic. In the static decision mechanism, the application code is partitioned at development time. The static partition has the advantage of low overhead when the parameters are calculated correctly. Kumar et al. [84] shows the static partitioning of code based on total energy consumption (communication energy and computation energy). They formulate the offloading of a program based on the trade-off between the communication cost and computation cost.

In the dynamic decision mechanism, the program decides at runtime by adapting to different runtime conditions. Approaches in [35, 39, 80, 101] relies on making offloading decisions in a dynamic environment (e.g. wireless interference, network failure). While dynamic decision mechanism has the advantage to overcome latency, the environment changes can cause additional problems. For example, the transmitted data may not reach the destination, or the data executed on the server will be lost when it has to be returned to the sender [44].

In this work, we have used both static and dynamic decision mechanism for code offloading to achieve energy-efficient hybrid mobile-cloud computing systems. The static decision mechanism is used in offline profiling to find efficient configurations of applications. The dynamic decision mechanism for offloading is used in online profiling, where the decisions are made at runtime based on a change in the environment.

2.5 Optimising Multi-Objective Problems

A multi-objective optimisation problem is a problem of finding a vector of decision variables, which satisfies constraints and optimises a vector function whose elements represents the objective functions [38]. These functions form a mathematical description of performance criteria which are usually in conflict with each other. Hence, the term "optimise" means finding such a solution which would give the values of all the objective functions acceptable to the decision maker.

Mathematically [38], multi-objective optimisation can be described as,

$$\begin{aligned} &min[f_1(x), f_2(x), ..., f_n(x)] \\ &x \in S, \end{aligned} \tag{2.1}$$

where $n > 1$ and S is the set of constrains defined below

$$S = \{x \in R^m : h(x) = 0, g(x) \geq 0\}\,, \tag{2.2}$$

As stated in [96], multi-objective optimisation originally grew out of three areas: *economic equilibrium and welfare theories*, *game theory*, and *pure mathematics*. In multi-objective problems, typically there exists no single global solution. Often the aim is to search for a set of solutions that all satisfy a predetermined definition of an optimum. In the following subsection, we explain the concept of *Pareto-optimality* that defines the optimal points.

2.5.1 Pareto Optimality

Pareto optimality is a state when a solution or a set of solutions in the solution space are achieved so that no other feasible solution reduces at least one objective function without increasing another one [21]. As in multiple-objective optimisation, there are more than one objective to be optimised simultaneously. During the process of optimisation, one of the challenging steps occurs when the objectives are conflicting with each other. There exists a natural trade-off between the objectives, which creates a set of optimal solutions using Pareto-optimal theory [38] [78]. These solutions are popularly known as Pareto-optimal solutions or non-dominated set of solutions [41]. As they are not dominated by any possible solution in the solution space, they could be the best solutions achievable. To represent a Pareto-optimal set or non-dominated solutions on a two-dimensional graph, an approximation front called Pareto front/Pareto frontier is used [38].

2.5.2 Multi-Objective Optimisation Algorithms

Over the past few decades, multi-objective optimisation (MOO) algorithms have been a subject of interest to researchers for solving various multi-objective optimisation problem, in which multiple objectives are treated simultaneously subject to a set of constraints [109]. To find the Pareto-optimal solution set of a MOO problem, methods such as mathematical programming and nature-inspired metaheuristics may be used. Based on mathematical programming, a MOO problem is *scalarized* to formulate a single-objective optimisation problem. The Pareto-optimal solution set of the single-objective optimisation problem is then treated the Pareto-optimal solutions to the MOO problem [74]. For example, linear weighted-sum, goal programming and epsilon-constraints are some of the methods that can be used for mathematical programming based scalarization.

Search-based Software Engineering Using Multi-Objective Optimisation Algorithms

Search-based software engineering (SBSE) is a sub-category of software engineering that includes all related work where search-based optimisation is applied [66]. SBSE has been successfully applied to certain areas such as project management [48], software testing [19], software effort estimation [99].

A search-based MOO problem is defined as; finding (or searching for) optimal or near to optimal solutions in a pool of candidate solutions. Harman et al. [65] stated two ingredients of SBSE, a suitable representation of the problem and definition of a fitness (or objective) function. Representation of the problem is the starting point and is followed by the process of search, which is guided by the fitness function that correlates a better and worse solution.

Praditwong et al. [105] used search-based approach to software module clustering. They considered two software engineering objectives of high cohesion and low coupling between different modules of a software system. They used two different multi-objective optimisation algorithms: 1) Maximum Cluster Approach (MCA) and 2) Equal-size Cluster Approach (ECA). The concept of Pareto-optimality was used as an assessment criterion to determine how good is the Pareto front achieved by the two approaches (MCA and ECA) against single objective formulation [45]. The single objective formulation was done because the objectives are often in conflict. Therefore, combining them into a single objective may result in a suboptimal solution. They used a test-bed of 17 real-world applications to evaluate the approaches. From the experimental results, it was concluded that the single objective formulation could not be used for finding a non-dominated Pareto frontier of optimal solutions. Also, the results were particularly compelling for their ECA multi-objective approach, which outperformed the MCA approach. Both the multi-objective algorithms were implemented using Evolutionary Algorithms. In the following subsection, we will discuss the evolutionary algorithms.

Evolutionary Algorithms

Nature inspired approach that is powered by heuristic search techniques; evolutionary algorithms are a collection of optimisation algorithms that successfully handle large, complex and multi-model search spaces [110]. Multi-objective optimisation problems are more often solved by using multi-objective evolutionary algorithms (MOEAs) [138] and swarm intelligence based optimisation algorithms (SIOAs) [135]. MOEAs aim for searching the Pareto-optimal set of solutions in a single run [74, 121].

As a subset of MOEAs, the multi-objective genetic algorithms (MOGAs), such as the non-dominated sorting genetic algorithm-II (NSGA-II) [42], have been particularly widely researched in the family of MOO algorithms [37], because they are capable of efficiently constructing an approximate PF. Traditionally, the direct search algorithms such as *Hill-climbing algorithm* have the problem of getting stuck at the local optima. To overcome such problem the *Genetic Algorithms* have the advantage.

Genetic Algorithms

Genetic Algorithms (GAs) form a branch of EAs [103]. They are bio-inspired, based on evolutionary theory and have been used for solving various optimisation problems, including MOPs. In GAs terminology, the search step to finding the PS involves the bio-inspired processes of initialisation, evaluation, selection, crossover, mutation, and replacement. Typically, GAs uses the following procedures [59]:

- The population composed of individuals (strings) are the encoded versions of the input rather than directly using the input.
- To guide the search GAs use an objective/fitness function rather than a derivative.
- The search mechanism in GAs is probabilistic rather deterministic.

In GAs terminology, a *generation* is the iteration of the search, where the quality of the individual is evaluated. After each iteration, a new generation of individuals is created by taking advantage of the fittest individuals of the previous generation. The process of GAs is illustrated in Figure 2.3.

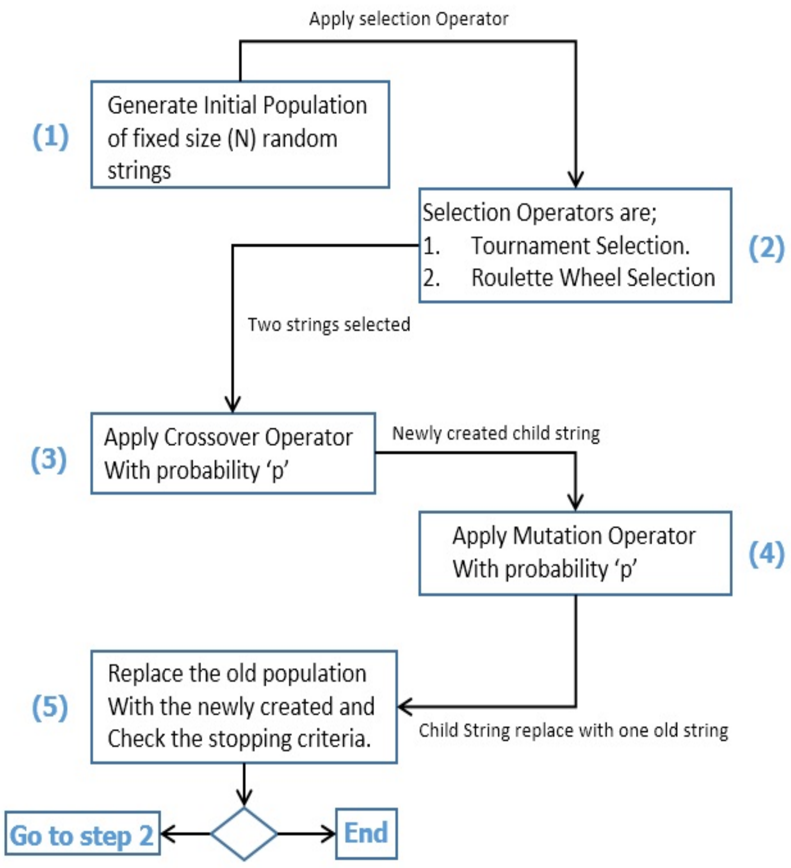

Apply selection Operator
(1)
Generate Initial Population of fixed size (N) random strings
Selection Operators are;
1. Tournament Selection.
2. Roulette Wheel Selection
(2)
Two strings selected
(3)
Apply Crossover Operator With probability 'p'
Newly created child string
Apply Mutation Operator With probability 'p'
(4)
(5)
Replace the old population With the newly created and Check the stopping criteria.
Child String replace with one old string
Go to step 2
End

FIGURE 2.3: The flow chart illustrates the procedure of bio-inspired Genetic Algorithms.

MOO to Reduce Battery Power Consumption in Mobile Devices

To reduce battery power consumption in mobile devices, MOO techniques can be used. In [89], the authors have presented a tool called *GEMMA*(**G**ui **E**nergy **M**ulti-objective opti**M**ization for **A**ndroid apps), which minimises the battery power consumption by using multi-objective optimisation, and Pareto-optimality. GEMMA produce a set of Pareto-optimal solutions that optimise the trade-off among the following three objectives.

1. Reducing energy consumption
2. Increasing contrast of the GUI
3. Making the GUI attractive

In this work, we are optimising two objectives using a hybrid mobile-cloud application framework to reduce battery power consumption. We will discuss HMC frameworks in Section 2.6. Like GEMMA, we use MOO techniques to optimise the trade-off between two objectives. Unlike GEMMA, we are using code-offloading to execute computationally-intensive tasks on the cloud, which minimise the battery power consumption of a mobile device.

2.5.3 Performance Measure of Multi-Objective Optimisation

To assess the outcome/result of different multi-objective optimisation algorithms, different performance indicators have been introduced [136]. Comparing the outcome of different MOEAs and evaluating it quantitatively is essential because it is usually an approximation of the PS. In order to measure the performance of the MOEAs, two goals are considered: 1) convergence of the true Pareto front and 2) distribution of approximated solutions. Generally, methods that assign each approximation set a vector of real numbers that reflect different aspects of quality are well accepted among researchers. The elements of the vector to represent the performance of MOEAs are called the unary quality indicators. Over the past few decades, many unary indicators have been introduced such as hyper-volume indicator [54, 29, 24] and attainment surface [92].

2.6 Hybrid Mobile-Cloud Frameworks

Hybrid mobile-cloud frameworks aim to integrate mobile applications into the cloud. The framework helps to partition a mobile application, and using code offloading executes resource-intensive modules of the application on the cloud. The design of HMC frameworks is generally based on achieving one or more than one particular objective(s), such as energy efficiency, storage, application execution time and bandwidth usage. Based on objectives, HMC frameworks can be classified into the following categories [77].

2.6.1 Frameworks to Improve Performance

This category includes HMC frameworks that are built with the primary objective to improve the performance of mobile applications. The aim of developing such frameworks is to execute computationally-intensive components of applications on the resource-rich cloud. Generally, the lightweight components of applications on a mobile device are offloaded to the cloud. This is due to the fact that it takes less time for the computation to complete on the cloud and return the result, as compared to the execution on mobile devices. Therefore, by minimising the execution time, the frameworks improve the overall performance of applications. For example, CloneCloud and DAvinCi are HMC frameworks that improve the performance of applications. They are discussed as follow.

1. *CloneCloud:* Proposed by Chun et al. [35], it offloads the computationally-intensive components of application code to a device clone operating in a computational cloud, to enhance application execution time. Therefore, improving the performance of the HMC applications. Using this framework, the mobile device and its clone on the cloud are needed to be synchronised all the times for consistent execution of HMC applications. Therefore, during code offloading, the application process on a mobile device enters a sleep state and transfers the process state to the clone. On the cloud side, a new process state is created and overlays the received information, followed by execution of the clone. When the execution completes, the process state of the clone's application is transferred to the mobile device. On the mobile device, the state is reintegrated into the HMC application which comes out of the sleep mode.

To partition an application using the CloneCloud framework, the source code is analysed first to generate a static flow control graph. The graph is important since it facilitates the process of partitioning. Based on the graph, the framework creates migrate-able points (partition). When the application executes and reaches a point, the framework migrates the process and the application's state to the clone. However, generating the migrate-able points are a challenging task as it can affect the overall performance gain of the application.

Chun et al. tested CloneCloud framework on a testbed containing three different tasks, i.e., virus scan, image search and behaviour profiling [35]. The results show that CloneCloud based applications gained 21.2% performance improvement in terms of execution time. Using CloneCloud has an advantage of when a mobile device (i.e. a smartphone) is lost or crashed, its data and applications can be recovered from the clone. However, the continual synchronisation between a mobile device and the clone on the cloud may be costly in terms of using battery power and network.

2. *DAvinCi:* A cloud-based framework for mobile robots in large environment [22]. DAvinCi (Distributed Agents with Collective Intelligence) is implemented around Hadoop-based clusters powered with ROS (Robot Operating System) as a messaging framework. It uses Software as a Service (SaaS) model of the cloud to share sensor data with other robots and also to offload computationally-intensive components to processing nodes. The primary objective is to improve the performance of parallelising the FastSLAM algorithm.

2.6.2 Frameworks to Reduce Energy Consumption

Hybrid mobile-cloud frameworks that are developed to reduce the energy consumption of mobile devices are designed to utilise cloud resources. Using code-offloading, such frameworks can achieve minimal energy consumption as the resource-intensive computational components are executed in the cloud. Therefore, applications use less power on mobile devices. To reduce energy consumption, two challenges can be faced as listed below [108].

- Finding the optimal condition that is suitable for offloading computation to the cloud.
- Factors that need to be addressed while offloading computation to the cloud.

In [84], the authors address the issues and provide a formula (2.3) that provides when offloading is useful in terms of reducing energy consumption.

$$P_{\text{c}} \times \frac{C}{M} - P_{\text{i}} \times \frac{C}{S} - P_{\text{tr}} \times \frac{D}{B} \tag{2.3}$$

Where:

$$\begin{aligned}
C &= \text{the number of computing instructions to be offloaded,}\\
M &= \text{the speed of mobile device (instructions/second),}\\
S &= \text{the speed of the cloud server (instructions/second),}\\
P_{\text{c}} &= \text{the mobile device power consumption (watts),}\\
P_{\text{i}} &= \text{the mobile device idle power consumption (watts),}\\
P_{\text{tr}} &= \text{the mobile device transmission power consumption (watts),}\\
D &= \text{the bytes of data to be exchanged and}\\
B &= \text{the network bandwidth.}
\end{aligned}$$

With P_{c}, P_{i}, P_{tr} being constant, if the above formula produces a positive number, offloading reduces energy consumption, otherwise not. However, when mobile devices use code offloading, the available network bandwidth B is also used. This suggests that offloading too often also increase the power consumption by mobile devices. Therefore, it becomes a concern to reduce energy consumption as well as the number of offloads.

2.6.3 Multi-objective Application Frameworks

The purpose of designing multi-objective frameworks for mobile-cloud computing is to achieve more than one objective simultaneously, mainly energy efficiency in parallel with performance or network usage of applications. As there exists a trade-off between the objectives, multi-objective frameworks aim to find compromise solutions between the objectives. Also, they are considered more effective to be used in MCC, as they support several objectives. Below we discuss multi-objective MC frameworks.

1. MAUI: Proposed by Cuervo et al. [39], MAUI maximises the potential for reducing energy consumption and improving the overall performance through fine-grained (method-level) code-offloading. It offloads the computation to the cloud, given that the offloading is effective in terms of improving performance and reducing energy consumption.

   MAUI decision mechanism (profiler) is built using Microsoft .NET runtime to analyse the energy consumption at runtime during local and remote execution of applications. It uses a history-based approach for predicting the execution time of methods both locally and remotely. Therefore, when the code-offloading is effective in terms of reducing the energy consumption of a mobile device, then it is used.

   One of the advantages of MAUI is its use of dynamic partitioning of applications, which reduce the burden on programmers. Moreover, MAUI targets fine-grained (method-level) code-offloading instead of a large block of code, which reduces network usage. On the weak sides, MAUI profiler executes on the mobile devices, which consumes extra processing power, memory and energy. It adds overhead on mobile devices.

2. ThinkAir [80]: Like MAUI, ThinkAir supports method-level partitioning of applications. It offloads the computationally-intensive methods of an application to a clone running in the cloud. The primary objective of offloading to the cloud is based on objectives such as minimum execution time, reducing energy consumption and the previous history kept by the ThinkAir's profiler. Moreover, ThinkAir achieves the desired *QoS* by executing multiple clones of a smartphone in parallel. Therefore, reducing delays and improving performance.

   The main advantage of ThinkAir is that it reduces energy consumption using code offloading while taking into account reducing execution delays by doing on-demand resource allocation and parallelism. The shortcoming of using ThinkAir is that the profiler executes on mobile devices. Therefore, incurs additional overhead by using processing power, memory and energy.

3. Cuckoo [76] is designed for Android-based applications that offload the computationally-intensive components to Java Virtual Machine (JVM) residing on the cloud. The objectives are to reduce the energy consumption of HMC applications as well as to improve their performance considering the execution time. The main advantage of Cuckoo is that it provides tools for developers to develop HMC applications easily. Moreover, its automatic mechanism to separate the offloadable components of an application from mobile device limited components is another big advantage. The shortcoming of Cuckoo is that it does not support asynchronous callbacks and state transfer from remote resources.

4. EECOF: Energy Efficient Computational Offloading Framework [115] is designed to offload lightweight components to the cloud. The objectives are: 1) minimising power consumption by offloading computationally-intensive components, and 2) minimising the overhead of runtime offloading components. However, the framework uses runtime profiling on the device, which uses devices resources.

5. In other studies, frameworks or offloading schemes for mobile-cloud applications were proposed based on achieving multi-objective. Deng et al. [43] and Guo et al. [64] proposed mobile-cloud frameworks that use code-offloading and are based on two objectives: minimising power consumption and execution time of mobile applications. Moreover, studies in [18, 33, 83, 84, 87, 132] are based on multi-objective that include: reducing applications energy consumption and communication power during computation offloading. For optimal task offloading Rahman et al. [107] design a smart city based Cloud robotic framework for minimising time and energy consumption of resources. However, in this work, a single robot is solely responsible for offloading decision making. Also, they did not consider the cost of energy consumption during communication.

2.6.4 Frameworks Based on Dynamic Offloading

The decision to offload to a nearby infrastructure or cloud is not always straight forward, particularly when conflicting objectives are considered. In such situations, objectives can only be achieved when the operating environment is suitable. Therefore, the computation offloading techniques for mobile-cloud computing has shifted from static to dynamic and context-aware in which the decision mechanism is built on *when* and *what* to offload [83]. In dynamic offloading, an intelligent decision mechanism determines whether offloading is required or not. When the environment is suitable for offloading, objectives such as battery power consumption, communication overhead and execution time are achieved.

Based on dynamic code offloading, researchers have proposed HMC frameworks. Gu et al. [62] proposed HMC framework for adaptive offloading of computationally-intensive modules. Gonzalo et al. [73] developed an adaptive offloading algorithm based on both the execution history of applications and the current system conditions. Naqvi et al. [101] proposed a multi-objective optimisation framework called (*MAsCOT*), which employs Probabilistic Graphic Models (PGM) for self-adaptive decision support for code offloading. Nakahara et al. [100] bi-objective optimisation framework (*CoSMOS*) analyse each optimisation parameters (energy consumption and execution time) separately using cost function and self-adaptive reinforcement. Further to improve runtime computation offloading decision mechanism, mobile-cloud frameworks based on self-awareness can be used. Self-awareness can benefit mobile-cloud, fog and edge computing by enabling it the remote nodes using MC architectures [106]. In [46], the authors have applied self-awareness to IoT hardware chips.

2.6.5 Middleware Based Frameworks

Hybrid mobile-cloud frameworks enable mobile applications to use the shared pool of resources provided by the cloud. Different cloud services provider may operate different platforms. To counter the problem of platform-independent accessibility in MCC, MC frameworks based on middleware are used. The middleware provides an abstraction between a mobile device and the cloud and controls every aspect of the communication in between. In [52], a mobile cloud middleware has been proposed, to perform data-intensive processing invocation from mobile devices, and to introduce the platform independence feature for the HMC applications. Other researchers [25, 118, 129] have proposed methods that are based on the middleware solution.

In our previous work [15], we used a simple middleware-based framework, which was based on Google's Firebase[2]. Using Firebase or another such kind of system as middleware comes with many problems that we had highlighted. They work in an event-driven fashion, which calls back to the thread that starts its on-event handler. For example, if the user interface (UI)/main thread is waiting to get the results from the cloud, the callback from the Firebase handler will be blocked due to the inherent characteristic of the Android system. In this work, we use a socket-based framework instead, which work in a suspend-offload-receive-resume fashion.

2.6.6 Cloud-enabled Frameworks for Mobile Robots

As we discussed in Section 2.1, cloud computing offers three types of service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In the recent past, Chen et al. [34] proposed Robot as a Service (RaaS) that is based on Service-Oriented Architecture (SOA) of cloud computing. RaaS facilitates the seamless integration of robot and embedded devices into Web and cloud computing environments. In RaaS, the integration of cloud with robot services is done using Microsoft Robotics Developer Studio (MRDS) and Visual Programming Language (VPL). Based on RaaS, many computationally-intensive robotics and automation systems applications such as robot navigation by performing SLAM in the cloud [11] and next-view planning for object recognition [102] can be achieved.

In [90], the authors proposed a comprehensive distributed cloud-enabled robotics framework. Apart from combining cloud and the robot networks, they also provide additional security features in their framework. Arumugam et al. [22] proposed distributed agents with collective intelligence framework. They showed a pool of heterogeneous robots works together in large environments. Their implementation of the framework was based on *ROS*[3], a Robot Operating System, that was used for sensor data collection

[2]`https://firebase.google.com/`
[3]`https://http://www.ros.org/`

and communication. In [124], cloud-based formation control of robots on the ground has been demonstrated. Therefore, improving the performance of mobile robots as well as providing cloud-based energy efficient alternative.

2.7 Insufficiencies in Available Approaches

In this Chapter, we discussed the background of research carried out and technologies built around mobile computing, cloud computing, Edge/fog computing, mobile-cloud computing, offloading techniques, multi-objective optimisation and evolutionary algorithms. We discussed the trends in the current state-of-the-art HMC frameworks using computation offloading and achieving one or more than one objective. The objectives mainly are to improve the performance (minimise execution time) and reduce energy consumption (minimise computation power and communication power) of applications developed for mobile devices. To achieve the objectives, static computation offloading can be used. However, due to changing environmental conditions such as weak signal strength, network link failure, network congestion, static offloading can cause latency and use more battery power (i.e., communication power for retransmission of dropped packets). In such cases dynamic offloading decision based on context-awareness, self-adaptation and self-awareness can be beneficial.

In this work, we propose a method to optimise two objectives in applications created for Android-based (i.e., smartphones and tablets) and Linux-based (i.e., Raspberry Pi controlled robots) mobile computing systems. We consider two objectives to optimise which are: battery power consumption and network usage. The battery power consumption is the total power required by an HMC application, including computation and communication power, and power required for other components to operate such as LCD and memory. The network usage is the wireless network available to the mobile-devices (i.e., WIFI). Minimising power and network usage creates an efficiency trade-off. We use a general purpose hybrid mobile-cloud framework that employs code offloading to optimise the efficiency trade-off.

We modularise the source code of mobile applications using annotations at the class-level, method-level and mix of class and method level granularity. We use a converter that identifies annotated modules in the applications source code and automatically converts them into offloadable modules. Our framework allows for both offline profiling and online profiling. Offline profiling is to search for efficient configurations that optimise the trade-off between battery power consumption and network usage. Our framework is based on self-adaptive and self-aware decision mechanism to make runtime decisions based on a change in the environment and change within the system itself to avoid network latency. By using online profiling, the framework switches between the efficient configurations to minimise battery power consumption, network usage and improve the performance of applications by avoiding network latency. Using a custom workflow, we use an exhaustive search algorithm to find efficient configurations of HMC applications. We also do more in-depth statistical-analysis to refine the final configuration set. In the next Chapter, we discuss the proposed method in more details.

Chapter 3

Modularisation of Applications Developed for Mobile Computing Systems

In this chapter, we discuss our method to achieve energy-efficient applications for mobile computing systems (i.e., a smartphone or a robot). In mobile devices, battery power is often one of the essential resources available. When computationally-intensive applications are executed on these battery-powered devices, they consume the available battery power quickly. To lower the battery power consumption, techniques such as *code-offloading* [97, 50] can be used in MC frameworks. They enable the HMC applications to execute remotely (i.e., on the cloud server endpoint) as well as on the mobile device. From a mobile device (i.e., a smartphone or a robot) point of view, the code-offloading is useful to minimise the device's battery power consumption with a trade-off of using the network and a constraint that it must not degrade the overall performance of the application.

The HMC application framework has a decision mechanism implemented that choose at runtime of the applications that which code units are to be offloaded to the cloud. The code units of mobile applications (e.g, classes or methods) are identified either during the development stage or after the developmental process is completed. This process is called modularisation. The identified code units (we called them modules) are made offloadable so that they can be executed remotely on the cloud server endpoint. The offloadable modules are usually composed of the computationally-intensive tasks of the

HMC applications. As these modules use the hardware components (i.e. CPU, RAM, WiFi or 3G/4G) of a mobile device, the available battery power is consumed when they perform the task. For example, as shown in Figure 3.1, an arbitrary Android-based HMC application. It has two offloadable classes (*Class1* and *Class2*) and four offloadable methods (*method1, method5, method6* and *method7*). The offloadable modules can be executed both locally and remotely (i.e., on the cloud server endpoint). The remote execution of the modules can reduce the battery power consumption of the application with a trade-off of using the available network.

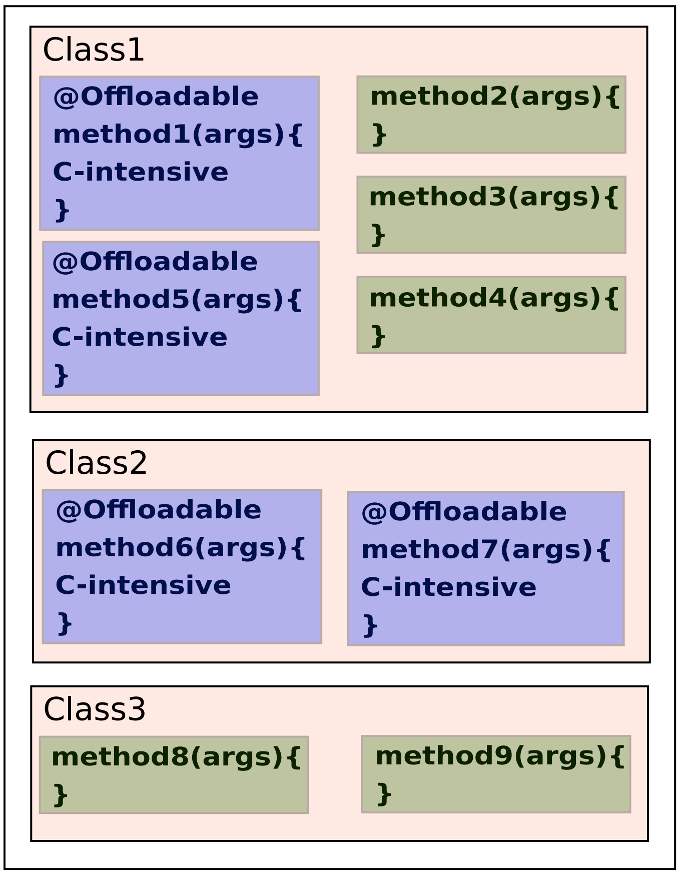


FIGURE 3.1: A high level view of a hybrid mobile-cloud (HMC) application consists of offloadable modules.

The decision mechanism of the MC Application framework, to execute which offloadable modules on the cloud and which on the device, is based on a *configuration*. We define a configuration as identifying all the offloadable modules of an HMC application and representing them in such a way that they can be executed either on a mobile device or on the cloud. To represent a configuration, we use a binary string of bits, where a zero represent that the mapped module will execute on a mobile device and one represent that the module will execute on the cloud. We create a range of different configurations for a mobile app or task, which is based on 1) granularity-level of an offloadable module, 2) execution of the offloadable modules across two endpoints (i.e., a mobile device and the cloud), and 3) the total number of the offloadable modules.

3.1 Granularity of Configurations

Granularity is the extent to which the HMC applications can be partitioned into different modules. The partitioning can be done at different levels of granularity: class-level, method-level, object-level, thread-level, task-level, component-level. In our case studies, we will be using mobile applications that are developed using Object-Oriented programming. Therefore, we will consider method-level and class-level partitioning of applications. The computationally-intensive tasks of the apps are, therefore, divided into the resultant partitioned components; we called them modules. The modules are composed of code units or machine instructions. They might be fine-grained (i.e., methods of classes) or coarse-grained (i.e., classes of an app). The fine-grained modules comprise of a chunk of code that might be computationally-intensive or is executed often during runtime of the app. In both cases, making the fine-grained modules offloadable and executing them remotely on the cloud can reduce power consumption. The coarse-grained modules are comprised of one or more fine-grained modules (methods). For example, as we can see in Figure 3.1, An HMC application has two offloadable coarse-grained modules and four fine-grained modules. One of the coarse-level modules (*Class3*) is not-offloadable, as it has no offloadable fine-grained modules.

We make the modules (i.e. classes and methods) offloadable at the code-level during the developmental stage by employing the MC application framework's APIs. During the runtime, the framework uses a configuration string to find the execution endpoint of an offloadable module. We create the configurations to represent the offloadable modules of an HMC application. The total number of distinct configurations for an app depends on the total number of its offloadable modules and their granularity levels.

3.2 Modular Configurations

A modular configuration (or simply a configuration) maps the offloadable modules to their execution points. It is created for executing a HMC application using the hybrid mobile-cloud framework. To map the offloadable modules of a HMC application to a configuration, the number at which a module execute during the application runtime is assigned its position (or index) in the configuration. A module may execute multiple times during runtime, but its position in the configuration is determined by the number at which it executes for the first time.

3.2.1 Representation of Configurations

We use a binary string to represent the configuration. Each digit (bit) of the string represents an offloadable module of the application. The state of a digit value signals the MC framework to execute its representative module either on a mobile device endpoint (0) or on the cloud server endpoint (1). Based on the total number of modules and their granularity level, we can obtain different types of configurations.

Class-level configuration

A configuration is a mapping of offloadable modules of an HMC application to a binary string. For creating class-level configurations, we consider the offloadable classes of the application as the modules. In a class-level configuration's binary string, each digit represents a class. The state value of a digit in the string guides the underlying HMC framework about the execution machine. In case the value is zero 0, the mapped class is executed on a mobile device. In case it is one 1, the mapped class is executed on the cloud server. For example, for an arbitrary HMC application, having four modules (as shown in Figure 3.2), a four digit ($n = 4$) binary configuration string would map the modules as 1010. In this configuration, the first digit (1) mapped the first offloadable module that will be executed on the cloud server as its state value is one (same applies to the rest). A combination of different configurations can be created, and the total number depends on the number of offloadable modules n. For $n = 4$, a set of class-level configurations is thus obtained and its cardinality would be $2^n = 16$.

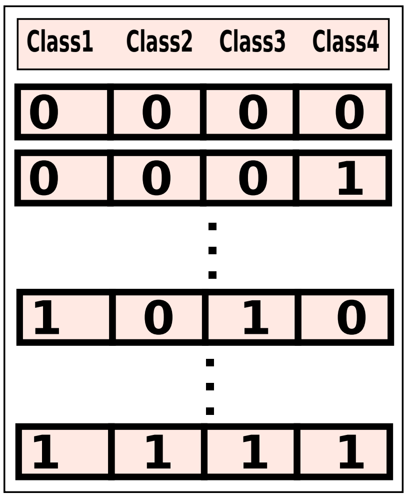


FIGURE 3.2: A class-level configuration set having all possible configurations for four offloadable classes of an application. The cardinality of the set is: $2^4 = 16$

Method-level configuration

Similar to the class-level configuration, a method-level configuration can also be created. In a method-level configuration, we aim to map the methods of a HMC application to the binary digits. The methods should be offloadable modules. The method-level configuration can also be represented as a binary string. For example, a method-level configuration, 10101010, from a set as shown in Figure 3.3. It maps eight ($n = 8$) modules of an app. To make the method-level configuration set, the cardinality of the set will be $2^8 = 256$. Each configuration will be a valid combination of mapping the methods.

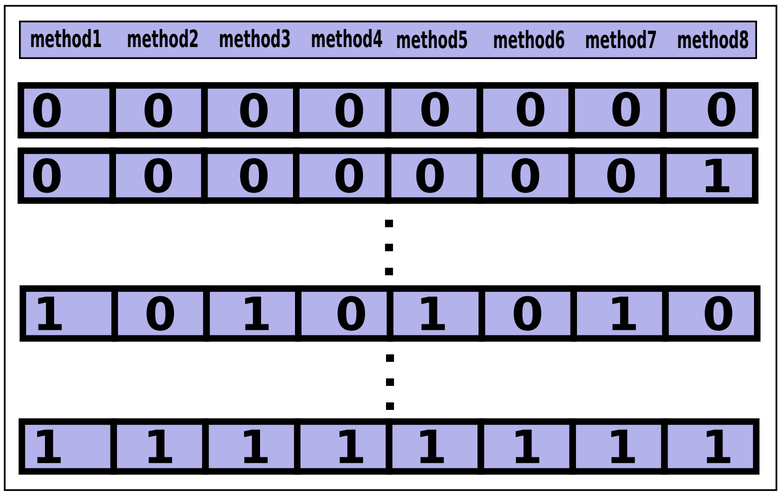


FIGURE 3.3: A method-level configuration set containing all possible combination of configurations for eight offloadable methods of an application.

Hybrid-level configuration

A *hybrid-level* configuration has a mixed granularity. To create a hybrid-level configuration, we aim to map the offloadable modules in a mixed combination of selected methods and classes. A binary string can also represent a hybrid-level configuration, i.e. $1010 : 01011010$, as shown in Figure 3.4. Unlike the class and method level configurations, the binary string is composed of two parts that are separated by a colon sign (:). The digits residing on its left side (first part) represent the offloadable classes, where zero (0) represent the class will be used coarse-grained and one (1) represents the class will be used fine-grained. The second part of the string (on the right side of the colon) is a combination of both classes and methods. The digits represent the mapped classes and methods.

The state value of a digit in the first part signals whether the mapped class will be executed as a coarse-grained (0) or as a fine-grained (1). The state of a digit in the second part describes whether the mapped offloadable module (a class or a method) will be executed on the mobile (0) or the cloud (1). If a class is represented as fine-grained in the first part then its offloadable methods would be mapped. If it is represented as coarse-grained then it will be mapped.

For example, let's assume an arbitrary hybrid-level configuration: $1010 : 01011010$. The first part of the configuration 1010, map four classes. Given that the first class has three methods and is mapped to be executed as fine-grained "1", all its three methods are mapped in the first three digits of the second part "010", where a zero represents executing the method on the mobile device and one is for executing on the cloud. To make a hybrid-level configuration set, the cardinality of the set will depend on the total number of offloadable classes and methods of the application.

To make a hybrid-level configuration set, the cardinality of the set will depend on the total number of offloadable classes and methods of the app.

3.2.2 Collapsible Configurations

We define a collapsible configuration as a configuration that can be collapsed into the same or a different granularity level configuration. In other words, two or more configurations are called collapsible if they are of different or same granularity level, and they

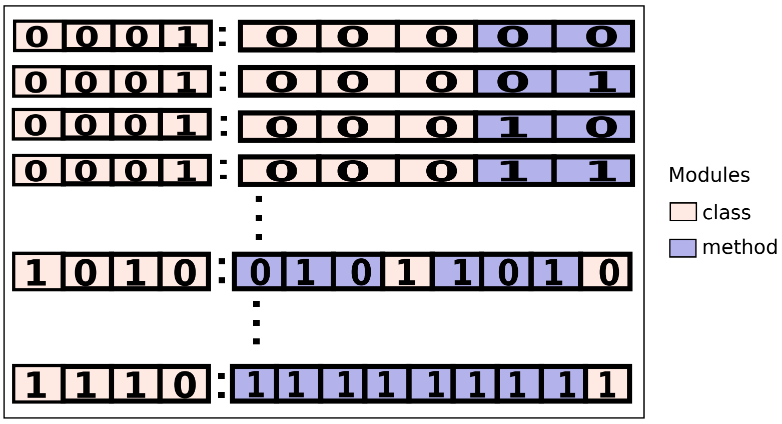


FIGURE 3.4: A hybrid-level configuration set containing all possible combination of configurations of an app.

map the same modules to be executed on the same endpoints. Collapsible configurations are identical because during the runtime the same modules are executed on either a mobile device of the cloud server.

As shown in Figure 3.5, a hybrid-level configuration, 1100 : 0001110, is identical to another hybrid-level configuration (0011 : 0111100) a method-level configuration (0001111100) and a class-level configuration (0110). In first configuration (1100 : 0001110) the first three digits (representing three methods of a class) on the left side of the colon are zero, which is same as the class-level digit in the second configurations (0011 : 0111100). In these collapsible configurations, we have assumed four classes. First and third class have three offloadable methods and second and fourth class have two offloadable methods. These configurations will execute the same modules on the similar endpoints no matter what the configuration level is. Even though collapsible representations are equivalent from a configuration perspective, they may still lead to different battery power consumption and network usage, due to implementation details. For example, finding the mapped modules for a hybrid-level configuration will go through many steps of (if-else) statements. On the other hand, for a class-level configuration, it will take relatively less number of comparisons.

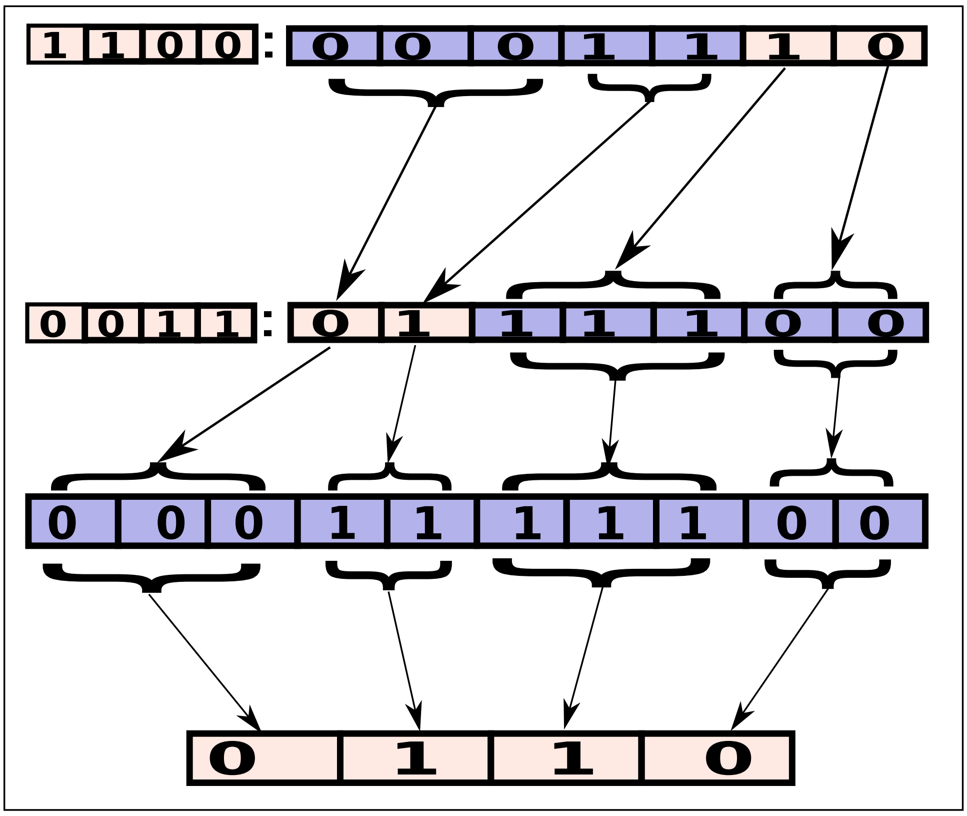


FIGURE 3.5: A set of four collapsible configurations. Each configuration is collapsible into another, where the execution endpoint of the offloadable modules are retained.

3.3 Hybrid Mobile-Cloud Application Framework

A mobile-cloud application framework provides the code-offloading API. We employ a simple MC application framework similar to the one used in [51]. During the development, the suitable modules (methods in classes) for offloading are annotated with (`@Cloud`). The offloadable modules should not be using any device limited library or components (i.e., GPS, Sensors, LCD, Input/Output). These modules might have computationally-intensive tasks. Annotating the modules with `@Cloud` is the only manual task that a developer is required to perform. These modules are converted automatically by a converter. We created two different versions of the framework: 1) Java-based, 2) Python-based. They are discussed in the following subsections.

3.3.1 Java-based Framework

The Java-based framework is used for the HMC applications developed for Android-based mobile devices. It implements the Java Socket API, which is used for code-offloading for remote execution of the offloadable modules on the cloud server endpoint. To use the framework, an annotated Android-based application is first converted to an HMC application. A converter tool, available on Github[1], can be used to convert the annotated methods of an application into offloadable modules. This tool generates two copies of an application: one for running on a mobile device (client) and the second is to execute on the cloud server endpoint.

We modified the converter so that it can inject some code into the landing (starting) activity of the converted HMC applications. This injected code is to check for the command-line arguments when the application is started. The configurations are passed to the HMC applications as command-line arguments. Depending on the configuration level, the digits set as 1 indicate: 1) all the offloadable methods of the classes will be executed on the cloud (class-level), 2) the offloadable methods of the classes will be executed on the cloud (method-level), 3) the offloadable classes or methods will be executed on the cloud (hybrid-level).

3.3.2 Python-based Framework

The Python-based implementation of the framework is general purpose, and here is used for HMC tasks performed by battery-powered robots. In all respects, it is similar to the Java-based version of the framework. It uses Python's socket library for communication between the client and the server and checks for the command-line configuration string during the start of a task. Using the Python-based version of the framework, we were able to execute a task performed by a battery-powered robot (3.4.2).

3.4 Case Studies

We will consider two different types of mobile computing systems, throughout this thesis, as a testbed: Android-based smartphones and mobile robots. We will be using the

[1] `https://github.com/huberflores/CodeOffloadingAnnotations`

framework and multi-granular configurations on the applications developed for these computing systems. In the following subsections, we introduce two HMC applications for Android-based devices and one HMC task performed by a mobile robot.

3.4.1 Android Applications

The first case study is targeting those applications that are created for Android-based devices (i.e., smartphones, tablets). These are battery powered mobile devices and are perfect candidates for applying our method of achieving energy-efficient HMC applications. Also, the Android OS is the most widely used operating system for mobile devices. It is based on the Linux kernel and developed by Google but later on by Open Handset Alliance (OHA). Its native language is Java, which is the officially supported language for Android development. Android OS is a stack of software components, which consists of five main layers:

1. Linux Kernal: The kernel is considered the heart of any OS. Android is built on the powerful Linux kernel, which provides basic system functionalities like memory management, process management, display, camera etc.
2. Libraries: The important libraries such as Webkit, SQLite and OpenGL sit on top of the Linux kernel.
3. Android Runtime: It consists of an optimised virtual machine for Android called *Dalvik Virtual Machine*. The Dalvik VM is where the applications run.
4. Android Framework: This layer provides higher-level services to applications such as activity manager, telephony manager, notification manager etc.
5. Applications: This is the top layer of the Android architecture. The developers write applications and install on this layer.

In the following sub-sections, we discuss two Android-based mobile applications that we will be using to apply our method.

ImageEffects

ImageEffects is a prototype Android application that we created. The idea was to use a test application similar to Instagram that does image processing and can also store the images on a remote file storage server. ImageEffects has built-in image filters to apply to an image that is taken by using the device's camera inside the app. After the filter is applied to the image successfully, it is then saved on a storage server. Mobile battery power is consumed when performing computationally-intensive tasks, i.e., image processing. The available WiFi/Data network is used for uploading/downloading images to/from the remote storage server. Also, battery power is consumed: 1) by the mobile device's WiFi chip during the uploading/downloading process, and 2) by the mobile device's LCD screen when displaying the application's UI.

The ImageEffects performs four distinct tasks. All the tasks are carried out in different modules (i.e., Java classes and methods) of the application. A sequence diagram is shown in Figure 3.6, which shows all the tasks and how they are performed. The first task is to apply a filter to an image, which is to change every pixel of the original image. The second task is to generate a thumbnail of the filtered image, which is scaling down the image to a lower resolution. The thumbnail is then set to be viewed on the application's User Interface (UI). The third task is to upload the image to the storage server and download it back. The fifth and last task is to calculate the hash codes of the original filtered image and the downloaded image and compare them to check that the downloaded image has not been changed or tampered. These tasks are executed in the background using the *AsyncTask* library of Android after input from the user.

The ImageEffects has a total number of 4 classes and there a total of 10 methods in these classes. They are suitable candidates for remote execution as they do not use mobile limited resources. The first and fourth class have three methods each and the second and third class have two methods each. Different granularity levels of configurations for ImageEffects can be created by following the approach discussed in section 3.1. For a total number of 4 class-level modules, $n = 4$, the cardinality of the class-level configuration set for ImageEffects will be $2^n = 16$. For 10 method-level modules, $n = 10$, the cardinality of the method-level configuration set will be $2^n = 1024$. Finally, the cardinality of the hybrid-level configuration set for the ImageEffects will be 2560. We have created

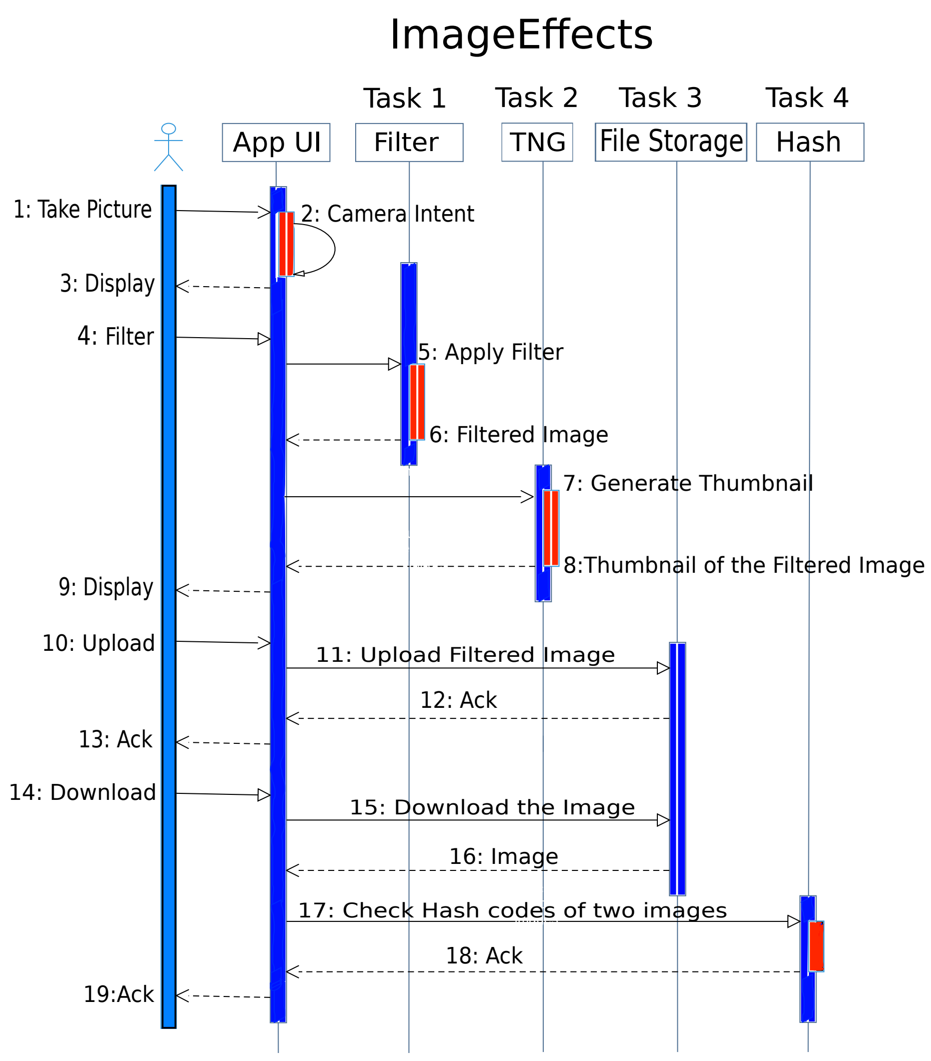


FIGURE 3.6: Sequence Diagram of the ImageEffects HMC application - a prototype and Instagram like the application we developed. The four distinct tasks of the application are performed in different granularity level modules, where each module can be executed either on a mobile device or the cloud server, based on a configuration.

a small Java-based tool, available on GitHub[2], which can be used to create all the valid hybrid-level configurations for a HMC application by providing the number of classes and methods.

To execute the ImageEffects, first, its methods are made offloadable. As discussed in Section 3.3, we used custom-made Java annotation to specify the methods in classes

[2]https://github.com/aamirakbar/Creating-Hybrid-Level-Configurations-for-MC-hybrid-Apps

that are comprised of the computationally-intensive tasks or are suitable for offloading. The annotation is placed during development time, before the methods, and is a part of the code-offloading API implemented in the Java-based MC applications framework (discussed in Section 3.3.1).

The offloadable modules (methods) are executed either on a mobile device or the cloud server when the application executes. The decision is based on the applied configuration. The efficiency of the configuration is then measured at the runtime. We will discuss how we measured the efficiency of the configurations in Chapter 4.

The total number of valid and possible configurations (combining all granularity levels) that can be created for ImageEffects is 3600. As mentioned earlier, there are total of four offloadable classes and ten methods in ImageEffects and their possible combinations ($2^n = 16, 2^n = 1024$) results in 1040 configurations. Adding the hybrid-level configurations (2560) of ImageEffects will make the total number of these configurations. Some of these configurations are collapsible and will end up in the collapsible sets, as discussed in Section 3.2.2. For example, a class-level configuration 0000 is collapsible into a method-level 0000000000, a hybrid-level $0001 : 00000$, another hybrid-level $0111 : 00000000$, and 14 other hybrid-level configurations. Similarly, a method-level configuration 0010110111 is only collapsible into a hybrid-level $1110 : 001011011$. In Chapter 4, we will discuss how we filter the collapsible sets for statistically-significant and non-dominated configurations. The filtered configurations will make the final configuration set.

Mather

Mather is an open-source Android application (available on GitHub[3]), which can be used for expression-based computations. In addition to basic arithmetic, Mather also supports complex mathematical expression evaluation, user-defined functions and matrices. Mather is based on the *Math.js*[4] library. When executed on an Android-based smartphone, it utilises the device's CPU for computation. The more complex the expression to evaluate, the more CPU utilisation and, therefore, more battery power consumption. To save battery power, the task of mathematical evaluations can be remotely executed on the cloud server using our Java-based framework. As the code-offloading comes with

[3]https://github.com/icasdri/Mather
[4]http://mathjs.org/index.html

the cost of network usage and power consumption of the WiFi/3G transmitter chip. Therefore, the framework will be used to find efficient configuration(s) that minimise the efficiency trade-off between power consumption and network usage.

We analysed the source-code of Mather to find the modules that are suitable for remote execution using code-offloading. Two of its classes (MathParser.java and MathItem.java), each having three methods, do all the mathematical evaluations and can be made offloadable. They are the most suitable fit we found for the remote execution.

We also found that Android's WebView is used in one of the methods, "`eval()`", which is run by the main UI thread. In the Android-based systems, when the main thread is running, a request from the cloud cannot be completed. For example, when `eval()` executes on a mobile device and call another method that is to be executed on the cloud according to the configuration. The result from the cloud would not be handled as the main thread will still be running and in a waiting mode. This will cause the application to enter into *ANR* (Application Not Responding), which is when the application is using the UI thread and waiting for response from the cloud server. Therefore, only one third (75%) of the configurations for Mather are feasible.

For 2 classes ($n = 2$), the cardinality of the class-level configuration set for Mather will be $2^n = 4$, in which only 3 are applicable. As in each of the classes, there are three offloadable methods. So, for 6 methods ($n = 6$) the cardinality of the method-level configuration set will be $2^n = 64$. The feasible method-level configurations are only 48. Finally, for two classes and six methods, we apply the valid 32 hybrid-level configurations.

Some of the configurations are collapsible and will end up in the collapsible configuration sets. For example, those configurations which run the modules only on a mobile device are collapsible and will form a set (i.e., $00, 000000, 01 : 0000, 10 : 0000$). Similarly, these configurations will end up in a same collapsible set, $111000, 01\!:\!1000, 10\!:\!1110, 10$.

In Chapter 4, we will discuss how to measure the efficiency of the configurations created for Mather and filter the statistically-significant and non-dominated configurations from the collapsible sets. The final configuration set will be created after the filtration, in which the efficient configurations will be picked for Mather.

3.4.2 Robotics

In the second case study, we will evaluate a task performed by a battery-powered and a Raspberry Pi controlled Thymio-2 robot as shown in Figure 3.7. In a mobile scenario, battery power is consumed by performing computationally-intensive tasks. To minimise the battery power consumption, remote execution of such tasks on the cloud can be done at the cost of using the network. We will be using our Python-based MC framework (discussed in Section 3.3.1) for the task performed by the robot to apply the configurations and then search for the energy-efficient configurations - those that optimise the trade-off between the battery power consumption and bandwidth usage. The optimisation will be discussed in Chapter 4.

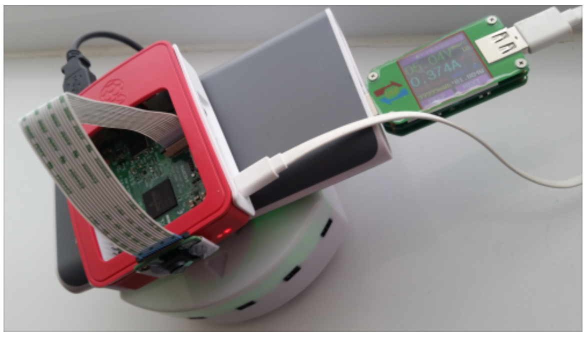


FIGURE 3.7: A battery powered and Raspberry Pi controlled Themio-II robot. The robot performs a foraging task. A portable Anker Power-bank powers the Raspberry Pi. The Robot has its battery, which is charged from the Raspberry Pi through a USB data cable. A digital multimeter placed in-line between the Power-bank and the Pi measures the power consumption of the Pi.

To evaluate our Python-based framework, we employed the work carried out by Heinerman et al. [68]. Using the API of the framework, we partitioned its code to make it executable across mobile robots and the cloud. In their work, a battery-powered Thymio-2 robot evaluates a foraging task online, which is to collect red coloured pucks in an arena and carry them to a blue shaded target area in the corner of the arena. A Raspberry Pi Linux-based system, which controls the robot, runs the code of the foraging task (written in Python language) as shown in Figure 3.7. The controller of the robot is a feed-forward neural network, which evolves on-the-fly as it performs the task. In the code, they have implemented a Python-based library, NEAT [119], to optimise an objective function that assesses the robot behaviour for some time.

The source code of the foraging task is available online on GitHub[5]. After analysing the code, we found a total of 4 classes that are suitable for the remote execution. These classes are from the NEAT library, which implements an evolutionary algorithm and a neural network. In these four classes, we found a total of 14 methods suitable for remote execution.

Next we create the configuration sets for the foraging task. For 4 classes ($n = 4$), the cardinality of the class-level configuration set is $2^n = 16$. For 14 methods ($n = 14$,) the cardinality of the method-level configuration set is $2^n = 16,384$. For two classes and fourteen methods, the cardinality of hybrid-level configuration set is $16,000$.

3.5 Summary of Contributions

In this chapter, we discussed partitioning the code units of mobile applications (developed using Object-Oriented programming paradigm) into different offloadable modules. We discussed how offloadable modules for two different computing systems (Android-based smartphones and Raspberry Pi controlled robots) could be represented using class-level, method-level and hybrid-level configurations. We also presented our general purpose hybrid mobile-cloud application framework for the two computing systems. The framework uses the code-offloading technique to execute offloadable modules of HMC applications remotely on the cloud.

Using the HMC application framework, an application's source code is modularised or partitioned during development time using a custom-made annotation before methods (Java and Python). These annotated methods are then converted into offloadable modules using the code-offloading API provided by the framework. During runtime, the framework decides whether to execute an offloadable module locally on the device or offload it to the cloud. The decision mechanism of the HMC application framework is based on a configuration. A configuration is a binary string that represents the offloadable modules and their executing endpoints. Based on the level of applications granularities, we consider a configuration one of three types: 1) class-level (coarse-grained), 2)

[5]`https://github.com/jvheinerman/NEATThymio`

method-level (fine-grained) and 3) hybrid-level - mix of coarse and fine grained. We create three levels of configuration sets for applications based on the configuration types. At the end of this chapter, we introduced test beds that consist of two Android-based applications (ImageEffects and Mather) and one robotic task (foraging). We discussed partitioning their source code, implementing the MC framework and creating their configuration sets. The offloadable modules of these HMC applications, when executed on the mobile devices, will consume the battery power. In order to save the battery power, they will be executed remotely on the cloud endpoint with a cost of using the network and power consumption of the transmitter chip.

As the following chapter will explore, the aim of using this method is to optimise the battery power consumption and network usage by 1) applications created for Android-based mobile devices, and 2) tasks performed by Rasberry Pi controlled and battery powered robots. In order to optimise the trade-off between minimising power consumption and bandwidth usage, we will be using Multi-Objective Optimisation to get Pareto-optimal configurations using offline profiling. These configurations are non-dominated by others concerning either of the two objectives: minimum power and minimum network usage. Executing the HMC applications with these configurations provide an alternative to achieve efficient hybrid mobile-cloud computing (HMCC) systems. In the next chapter, we will discuss how to create an experimental setup to do offline profiling. Specifically, we will discuss how to: 1) measure the efficiency of the configurations, 2) do statistical tests and create the final configuration sets, and 3) find the Pareto-optimal configurations.

Chapter 4

Multi-Objective Optimisation of Hybrid Mobile-Cloud Applications

In this chapter, we will discuss a workflow to achieve energy-efficient HMC applications developed for Android-based and Linux-based mobile computing systems. The workflow aims to search for an efficient trade-off between power and network usage by the HMC applications. Using the method discussed in Chapter 3, the HMC applications are executed across mobile and cloud endpoints. By instrumenting the applications and using offline profiling, the efficiency of the configurations is measured.

The workflow is employed to find the Pareto-optimal configurations for the two HMC Android-based applications (ImageEffects and Mather) and the robotic task (Foraging). The Pareto-optimal configurations are those that are non-dominated by others, by considering the two objectives: 1) minimising battery power consumption and 2) minimising network usage.

4.1 Evaluating Configurations

The efficiency of a configuration is the recorded power and network usage of a configuration when it is used for a single run of the application using offline profiling. However, as we will see later in offline profiling, we will execute an HMC app multiple times with a configuration because the objective functions are stochastic. When an HMC application executes, battery power and network usage are recorded for the whole execution time. The two objectives of minimising the battery power consumption and network usage are conflicting. Therefore, they form an efficiency trade-off [15]. The recorded efficiency of each configuration is stored for later analysis. In below subsections, we discuss how we measure the efficiency of configuration created for the HMC applications.

4.1.1 Battery Power Consumption

Mobile devices operate on a limited supply of power available from the battery. Therefore, power consumption should be used carefully [93]. The offloadable modules of an HMC application, when executing on a mobile device, use the device components for computing. The components draw power from the battery to operate. We denote the computation power as: P^{C}. Alternatively, the modules can be executed remotely on the cloud. As stated in [47, 88], an offloadable module executes on the cloud in three phases: 1) the transmitter (RF) sending phase, 2) the cloud computing phase, 3) the transmitter (RF) receiving phase. We assume that the power consumption due to communication (RF sending and RF receiving) is: P^{RF}. The total power consumption, P^{T}, for a configuration can be found as follow:

$$P^{\mathrm{T}}{}_{(watts)} = P^{\mathrm{C}}{}_{(watts)} + P^{\mathrm{RF}}{}_{(watts)} \tag{4.1}$$

When no code-offloading is used, the communication power (P^{RF}) will be zero, and the total power consumption (P^{T}) will be equal to the total computing power (P^{C}).

Measuring battery power consumption

Battery power is consumed when an HMC application is executed on a battery-powered mobile device. Power is required for different components of the system to work, i.e., CPU, WiFi. To measure the total power consumption of an application as in Equation 4.1, we have employed different ways for Android-based mobile devices and Raspberry Pi-controlled robots.

Android-based systems

For the Android-based systems, we created a separate dedicated application, which we called *Monitor*, that runs on the device and outputs the total power consumption after a single run of the targeted HMC applications. Since it is important to measure the total power consumed by an individual HMC application, we have integrated an open-source power measuring tool called *PowerTutor* - available on GitHub[1]. PowerTutor is based on component power management and activity state introspection [133]. To estimate the power consumption of the components, PowerTutor uses PowerBooter; which is an automated power model construction technique that uses built-in battery voltage sensors and knowledge of battery discharge behaviour to monitor power consumption while explicitly controlling the power management and activity states of individual components. These components include CPU, WiFi, 3G, GPS, LCD and audio interface. Also, PowerTutor considers the energy consumption of each application to be independent. In other words, PowerTutor assumes that, i.e., app A consumes the same amount of energy with or without app B running. In this way, power consumption is measured (based on statistics) for each component and each UID/application. By embedding the PowerTutor in our Monitor application, the total power consumption (as stated in Equation 4.1) is measured during the runtime of an Android-based HMC application.

Python-based tasks for robots

The Raspberry Pi models have no inbuilt current or voltage sensors that could be used for monitoring its current draw, or battery supply. Therefore, an extra hardware device such as a multimeter or an onboard electronic shunt must be used [40, 49] for measuring total power delivered to the device. We used a digital multimeter that is placed inline between a power bank and the Pi as shown in Figure 3.7. The type of multimeter we used can measure the power consumption of the Pi every seconds and also is able to send the recorded readings through a Bluetooth connection - UM24C[2]. The readings from the multimeter showed that when the Pi starts executing a task, more power is drawn from the battery and the consumption increases. Therefore to find out the total power consumption during the foraging task, we first measure the total power consumption of the Pi for 5 seconds in the idle state. When the foraging task starts, we measure the total power consumption for the whole run of the task. In the end, we subtract the power

[1]`https://github.com/msg555/PowerTutor`

[2]`https://www.aliexpress.com/item/RD-UM24-UM24C-for-APP-USB-2-0-LCD-Display-Voltmeter-amme`
`32845522857.html`

consumed at the idle state from the power consumed when the task was running. This gives us an estimated power consumption of a single run of the task.

4.1.2 Network Usage

Network bandwidth is used when the MC framework is using code-offloading. However, code-offloading can be inefficient if an HMC application relies too much on it. It will use maximum network bandwidth as well as consuming battery power (P^{RF}). The total network usage for a configuration is recorded as total bytes transmitted (Tx) and received (Rx) between a mobile device and the cloud when an HMC application is executed. Also, network latency is a critical measurement in code-offloading. If packets are dropping due to low signal strength or there is congestion in the network, then the performance of the HMC application will be degraded.

Measuring Network Usage

Network bandwidth is used when an HMC application executes its modules remotely on the cloud server endpoint. To measure the total network bandwidth usage, we have employed different ways for Android-based mobile devices and Raspberry Pi-controlled robots. It should be noted that as we will be doing offline profiling to find the efficiency of configurations of HMC applications and the foraging task using a lab environment, network latency in such case is observed to be negligible. Therefore, we do not measure network latency in offline profiling.

Android-based systems

For Android-based smartphones, similar to measuring power consumption, network usage is also measured by the Monitor application. The Monitor implements a built-in Android Library, *android.net.TrafficStats*, for recording data-sent Tx and data-received Rx. As the measurements are recorded against the `UIDs` of all applications running on a smartphone, we can extract the recorded network usage measurement of a targeted HMC application by providing its `UID` in the Monitor application.

Python-based tasks for robots

For measuring network usage in Linux-based computing systems like Raspberry Pi, a range of command-line based packet analysing tools can be used. For instance, *TCP-dump*[3] allows the user to truncate and view TCP/IP and other packets transmitted/received over a network to which the computer is connected. Similarly, *NetHogs*[4] is a small, very useful tool to monitor network traffic by process-level. It is feature rich, straightforward to use and can be easily installed on Linux machines. NetHogs also makes it easy to identify programs that occupy an important portion of the available bandwidth.

Since all of these packet analysing tools can be used to measure the bandwidth usage of a mobile device, we choose *Tshark*[5], which is a command-line network packet analyzer tool. It capture the live network traffic and can be used from inside a Python script. We were able to measure the total number data transferred between the mobile robot and the cloud with Tshark by embedding it in a python script, and using the script to capture only the data traffic of the targeted foraging task. After each run of the task, the script gives us the total network bandwidth used during the runtime of the foraging task.

4.2 Offline Profiling

In offline profiling, we instrument the HMC applications to search their energy-efficient configurations using multi-objective optimisation. For this purpose, we wrote a Python-based script, which implements an exhaustive search algorithm that iterates through all the possible configurations. As we discussed in Section 3.4, the total number of configurations for ImageEffects are 3600, for Mather are 83 and for the foraging task are $32,400$.

The script runs on a PC and executes the HMC applications on a smartphone and the foraging task on the Raspberry Pi. One execution of the applications is with one configuration from their respective sets. For the whole runtime, the total power consumption, total network usage and total execution time are measured as discussed in Section 4.1. Also, executing the HMC applications manually and repeatedly with all of these configurations is time-consuming and practically not feasible. Therefore, an automatic process

[3]`http://www.tcpdump.org/tcpdump_man.html`
[4]`http://nethogs.sourceforge.net/`
[5]`https://www.wireshark.org/docs/man-pages/tshark.html`

is required that executes the apps repeatedly and each time with a new configuration. In the following subsections, we discuss how to automate the executions of the HMC applications.

4.2.1 Executing Android-based HMC Applications

The Android-based HMC applications (ImageEffects and Mather) are automatically executed using Python-based script. We used the exhaustive search algorithm, shown in Algorithm 1, to measure the efficiency of all the configurations. The algorithm repeatedly executes the apps in its outer while loop, each time with a new configuration. The Python script uses an open-source library called *AndroidViewClient*[6] to interact with the HMC application on the smartphone, which is connected to the PC via a USB cable. This library provides higher-level operations and the ability to obtain a tree of Android UI Views present at any given moment on the device or emulator screen and performs operations on it. Alternatively, Android's library *MonkeyRunner* can also be used for the same purpose.

An exhaustive search algorithm is pre-provided with all the configurations (S) for both HMC applications (ImageEffects and Mather). In each iteration, the Monitor application (discussed in 4.1.1 along with an HMC application is launched. A configuration is passed to them as a command-line argument. During runtime, the script sends inputs (commands, parameters and touch events) to the views of both launched applications. After the HMC application completes a single run, it is programmed to terminate. The script using the `ps` command finds the HMC application is terminated. The Monitor is then stopped by sending it the stopping command, and the recorded measurements from the Monitor for the current configuration are passed to the PC. This process is repeated until all the configurations are exhaustively applied to find their efficiency.

The Monitor app measures: the total power consumption of all the components (WiFi, CPU and LCD) the HMC application is using, the total network bandwidth usage ($Tx + Rx$) and the execution time of the app for the current configuration. When the automation script stops the Monitor app, a string having all the recorded measurements ($config + power + Tx + Rx + data + runtime$) for the current configuration is created.

[6] `https://github.com/dtmilano/AndroidViewClient/wiki`

It is then sent to the PC over a socket connection, which is then parsed and the recorded values are stored in a SQL database on the PC.

Algorithm 1 Exhaustive Search Algorithm Executes MC Hybrid Apps with Configurations and Measures Efficeincy of the Configurations

```
1:  $device \leftarrow connectToDevice()$
2:  if $device == null$ then
3:      return
4:  end if
5:  $S \leftarrow ConfigurationSet$
6:  $count \leftarrow 1$
7:  while $count \leq n$ do                                ▷ n = total number of configurations in S
8:      device.shell(Monitor App, $S[count]$)
9:      device.shell(HMC App ( $\alpha$), $S[count]$)
10:     while true do
11:         app ← device.shell(ps | grip  $\alpha$)                      ▷ $\alpha$ = HMC app
12:         if app == null then
13:             break
14:         end if
15:     end while
16:     device.touch(x,y)                                   ▷ Stop the Monitor app
17:     count ← $count + 1$
18: end while
```

4.2.2 Executing Python-based HMC Robotic Task

For the HMC foraging task performed by the Raspberry Pi controlled Thymio robot, we wrote another automation Python-based script (similar to the one we used for Android-based systems). The core of the script is composed of an exhaustive search algorithm, which goes through all the $32,400$ configurations of the task. The script runs on the PC and executes the foraging task using SSH to the Raspberry Pi. Using the exhaustive search algorithm, the task is executed repeatedly and each time with a new configuration. The battery power consumption is recorded using the in-line digital multimeter. The network usage is recorded using Tshark, as discussed in Section 4.1. The recorded battery power consumption, network usage and execution time are stored for later usage.

4.3 Finding Statistically Significant Configurations

Statistical test aims to uncover a significant difference between samples (we called a measured efficiency of a configuration after a single execution of the HMC application as a *sample*). In order to uncover the difference between the the power and data usage of collapsible configurations, we will do the statistical tests. Based on the result of the test, we will either accept or reject the null hypothesis. As discussed in Section 3.2.2, collapsible configurations are those that map same modules but having different granularity level.

During the statistical test, we will compare power and data measurements of individual runs of HMC applications. This is required because the collapsible configurations may lead to different battery consumption and network usage, due to their implementation as discussed in Section 3.2.2. For sample size, say "n", the efficiency of a configuration is measured "n" times. In this work we will execute the the HMC applications for multiple times with a each configuration, therefore, we will get a sample size of the number of times the application is executed.

The sample size is important because a larger sample size will give us more confidence over any expected difference, given the noise in the system. However, as there exist outliers of measured power or network usage of the executions, their presence leads to substantial distortions of parameter and statistic estimates when using statistical tests. Therefore, we first eliminate the outliers from the samples before using the statistical tests.

4.3.1 Outlier Elimination

We noticed during manual analysis of the data, generated during offline profiling (discussed in Section 4.2), that for some configurations the power consumption values, of some samples deviate markedly from others. Therefore, to eliminate such outliers, we do outlier elimination. Many outlier tests have been proposed [127]. Some of the more commonly used tests are the Grubbs' test, Tiejen-More test, Generalized Extreme Studentized Deviate (ESD) Test. We wrote a Java-based tool that implements Grubbs' test. It accesses all the "n" samples of the efficiency of the configurations and applies Grubbs' test repeatedly until all the deviated power consumption values are removed. It then takes new measurements to replace the eliminated ones in the samples. Now that we have the efficiency of all the configurations recorded with no outliers, we can carry out statistical tests. This will filter the statistically significant configurations in the collapsible sets.

4.3.2 Hypothesis Test

Finding statistically significant configurations is important because in the collapsible sets (discussed in Section 3.2.2) any observed difference between the efficiency of any two configurations may be due to other factors. In other words, the efficiency of any two collapsible configurations happened to be different due to some uncontrolled variables. For example, the operating system was doing system related tasks, and the HMC application took more computation time, or the data packets were dropping due to congestion in the network. Also, it might be due to any unexpected experimental error. As the uncontrolled variables cannot be avoided entirely so, due to their presence, to select efficient configurations hypothesis testing becomes essential.

A hypothesis cannot be proved, but can only be accepted/rejected based on a statistical test result. So, the test will either accept or reject a null hypothesis. If the probability value of the test for two configurations is less than 0.5%, the null-hypothesis is then rejected. This shows that the configurations are statistically significant. Therefore, to select the statistically significant configuration, the null-hypothesis should be rejected

We establish the null-hypothesis by assuming that any two configurations in the collapsible sets have no real difference in terms of their recorded power consumption and bandwidth usage values. The difference in the means of power and bandwidth happened merely due to the uncontrolled variables. The measured power consumption data of the samples of the configurations is not normally distributed. Therefore, non-parametric hypothesis tests are best to apply to such data. We created a Java-based tool that iterates through all the collapsible sets and applies Wilcoxon rank-sum test [67] (a non-parametric statistical hypothesis test) to the configurations in each set. The tool stores the statistically significant configurations from the sets. When the tests are completed, the tool then searches the non-dominated configurations in the collapsible sets, as discussed in the following section.

4.4 Filtering the Configurations

This is the final step in our workflow. To achieve efficient HMC applications, we use a filter only to pick (if present) the non-dominated configuration(s) along with their statistically significant configuration(s) in each of the collapsible sets. The rest of the configurations in all the collapsible sets are removed since they are dominated and are not statistically significant.

4.4.1 Selecting Non-Dominated Configurations

A collapsible set has one or more than one non-dominated configuration(s). They are more efficient than others in terms of minimum power consumption or network usage. The Java-based tool (discussed in Section 4.3.2) finds them and stores them in a final configuration set. This set is composed of: 1) statistically significant and non-dominated configurations in the collapsible sets, 2) non-collapsible configurations, which were not part of collapsible sets. The efficient configurations, for the Android-based HMC applications (ImageEffects and Mather) and the foraging task performed by the robot, are then searched in their final configuration sets.

4.4.2 Pareto Efficient Configurations in the Final Set

The final configuration set is created that has a mix of configurations of the three granularity levels. The non-dominated configurations in the final set are the Pareto-optimal configurations, making a Pareto-front as shown in Figure 4.1. These configurations are superior to the others when both of the two objectives are considered [117]. Also, these configurations optimise the efficiency trade-off and provide efficient alternatives to the HMC application in terms of battery power consumption and network usage.

4.5 Case Studies

In this section, we explain the experiments conducted and their obtained results. To achieve efficient hybrid mobile-cloud computing (HMCC) systems, we discussed our technique in Chapter 3. The applications selected as a test-bed for the experiments were introduced in Section 3.4. In this chapter, we established a workflow, that we use for

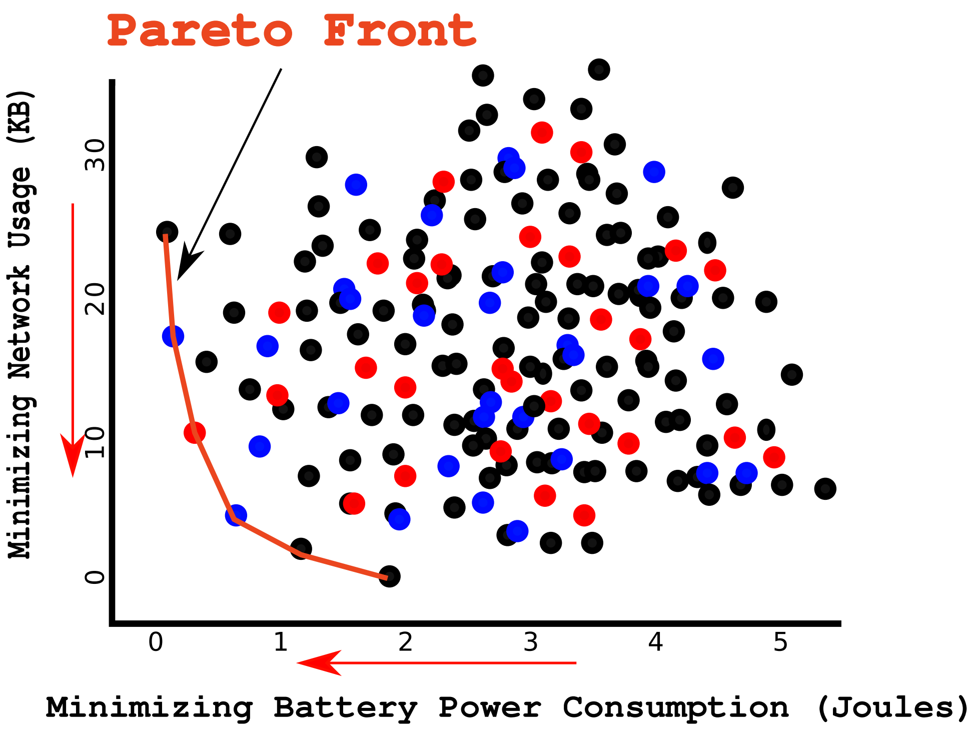


FIGURE 4.1: The efficiency of configurations are plotted in this graph. The configurations making the Pareto front are the Pareto efficient configurations. The dots represent class-level, method-level and hybrid-level configurations in three different colours.

multi-objective optimisation of the configurations for the HMC applications. We follow the workflow to carry out the experiments in the following subsections.

4.5.1 Experimental Setup

We created a lab environment to conduct the experiments, which was composed of the following entities.

1. Two endpoints, one for execution of the mobile application and one for the cloud application.

2. A desktop PC, which runs: 1) The automation scripts (discussed in Section 4.2) created for offline profiling of Android-based HMC applications and robotic foraging task. 2) The image storing server program for ImageEffects (written in Java). 3) A Java-based server program that receives the measurements from the Monitor app and stores them in a SQL database. 4) A Python-based program that implements *Tshark* for measuring network usage during code-offloading use by the foraging

task. 5) A Python-based program to communicate with the digital multimeter and measures the recorded battery power consumption of the robot.

3. A Small Office or Home Office (SoHo) network which connects the PC and the two endpoints.

For the Android-based HMC applications (ImageEffects and Mather), we used an Android-based smartphone "Motorola Moto G4" for the mobile endpoint. For the cloud endpoint, we set-up an Android-x86 virtual environment on the PC. The HMC applications and the Monitor application were installed on the endpoints. To send input commands or clicks to the views of the applications via ADB (Android Debug Bridge), the smartphone was connected to the PC through a USB cable. We execute the automation script on the PC, which runs the exhaustive algorithm to find the efficiency of all the configurations for the HMC applications. To get n samples of a configuration c, the script executes each of the HMC applications for a n number of times with the configuration c.

The Python-based script for the HMC foraging task (mobile version) was installed on a battery powered Raspberry Pi device, which controls a Thymio robot. We executed the server version of the task on the PC. To use the code offloading, both endpoints were able to communicate with each other using the SoHo network.

4.5.2 Understanding the Framework Used for Android Applications

Before we jump into the results obtained for the ImageEffects and Mather, it is important first to understand how the underlying mobile-cloud framework behaves with different configurations of the same application. For this purpose, We created a prototype Android-based HMC application (we call it $Proto$). The Proto application is build to use the HMC framework. It consists of two Java classes, where each has two offloadable methods. Each method evaluates a different and randomly selected arithmetic expression. These expressions are: 1) Given a number x, method "A" finds factors of all the numbers from 1 to x, 2) Given a number x, method "B" finds the factorial of x, 3) Given a number x, method "C" finds the multiplication of: $x * x$ matrix - having randomly generated numbers from 1 to 100, 4) Given a number x, method "D" evaluates the expression:

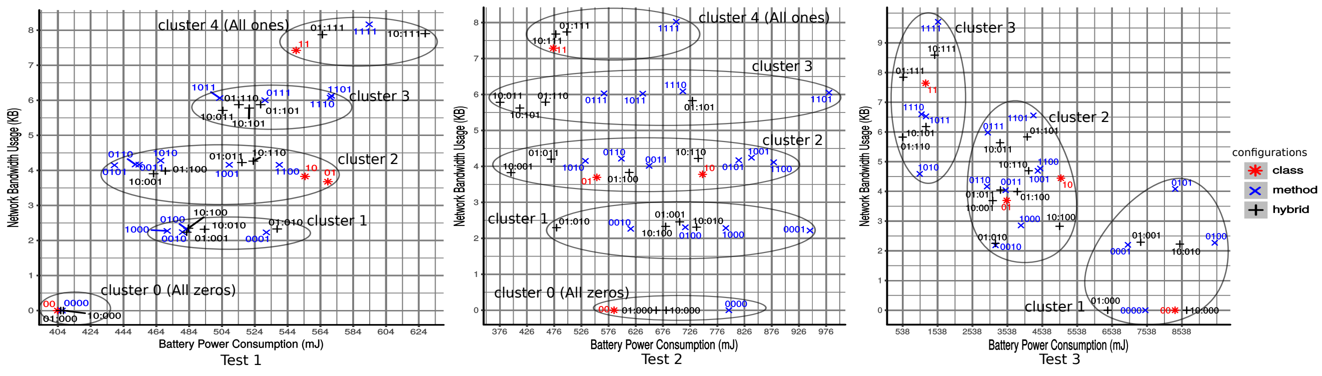
$tan(tan(tan(x)))$. Based on the three levels of granularity (discussed in Section 3.1), a total of 36 configurations for the Proto app were obtained.

We performed three tests with the Proto app. The computation level of the app was increased subsequently in each test, which was achieved by increasing the input value of parameter x in each method. In all the three tests, we obtained 100 samples of the configurations and then filtered the configurations to obtain the final set. The statistically significant and non-dominated configurations in the collapsible sets and non-collapsible configurations for the Proto app, for all the three tests, are plotted on 2D graphs in Figure 4.2. We produced two different dot-graphs (Figure 4.2a and Figure 4.2b) of the tests, in which the configurations of the Proto app before and after applying the filter are respectively shown. The data points on the graph are the configurations, where the x-axis value is the mean of the total power consumption and the y-axis value is the mean of the network usage. We analyse the results in the following subsection.

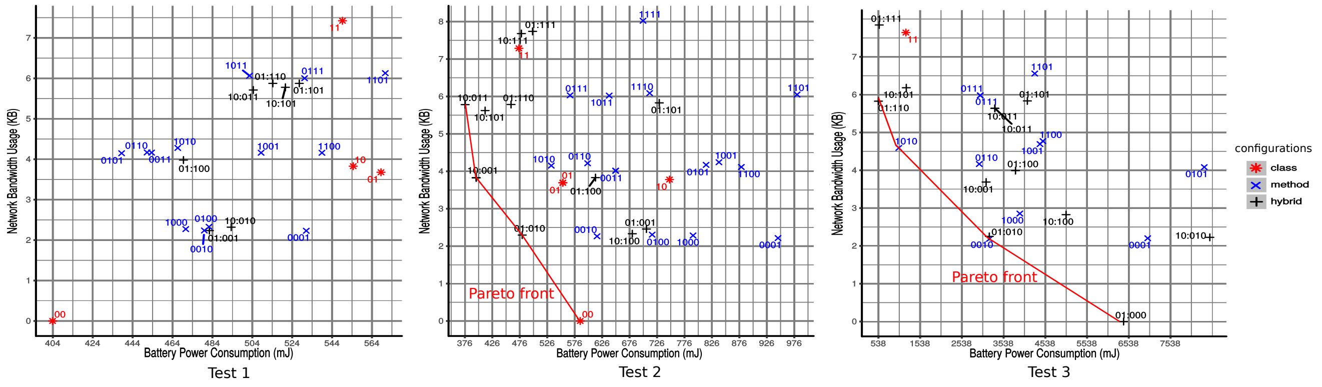
While the computation level of the Proto app in each test was increasing subsequently, the network usage of the app using code-offloading was the same. After the offline profiling of the Proto app, we have observed the following features from the results that reflect the behaviour of the framework.

All-zero and all-one configurations

These are the configurations with which an HMC application executes all the offloadable modules either on a mobile device (all-zero) or offloads (all-one) to the cloud. In all-zero configurations, the battery power is consumed as the modules execute on a mobile device and there is no network usage. The all-one configurations come with the cost of using the network and consuming the battery power by the transmitter chip. We can see in Figure 4.2a, when the computation level of the Proto app was low (test-1), the all-zero configurations were more efficient. As the computation level of the application was increasing (test-2 and test-3), the all-one configurations were turning out to be more energy-efficient than all-zeros with the cost of maximum network usage. The all-zero configurations can be seen in cluster-0 of test-1 and test-2, and cluster-1 of test-3. The all-one configurations can be spotted in cluster-4 of test-1 and test-2, and cluster-3 of test-3.



(a)



(b)

FIGURE 4.2: Plots showing the results obtained from three different profiling tests. The tests were carried out with a prototype Android-based application (Proto) to understand the underlying mobile-cloud application framework. The computation level of the app increases in each test subsequently. (a) All configurations of the app forming in clusters. We can see that the configurations are turning into vertical shape clusters from horizontal after increasing the computation level of the app. (b) The filtered configurations having the Pareto efficient configurations making the Pareto front.

Clusters of configurations

We can see in Figure 4.2a, the configurations of the Proto app formed into different clusters after their efficiency was measured. The clusters are formed based on the amount of battery power consumption and network usage. The configurations that have nearly the same network usage formed into horizontal type clusters. These configurations have the same number of modules mapped as one, which is to use code-offloading. As shown in Figure 4.2a, we can see four clusters of configurations in test-1 and test-2 that are numbered on the bases of the number of modules using code-offloading.

When the computation level was low (test-1), the configurations in the clusters were near to each other. When the computation level was increased (test-2), the distance between the configurations in the clusters also increased. This is because the modules that were doing complex computation consumed more battery power and moved to the right of their clusters. This behaviour is more prominent in test-2 (Figure 4.2a), where the configurations inside the clusters are more expanded.

When the computation level was maximum (test-3), the clusters of configurations become more prominent in vertical form rather than the horizontal form as shown in Figure 4.2a. Those configurations that consumed more battery power are now in the right cluster (cluster-3). This is because the modules they mapped were more computationally-intensive. As the third method of the Proto app is $x * x$ matrix multiplication, which is the most computationally-intensive method. Therefore the configurations with the third digit set to zero landed into cluster-3. In cluster-1, the configurations that consumed less battery power are present. These configurations offloaded the computationally-intensive modules to the cloud. Therefore, they consumed less battery power with the cost of using more network.

Pareto-optimal configurations

As discussed in Section 4.4.2, the final configuration set for the Proto app for all the three tests are created. This set contains the statistically significant and non-dominated configurations in the collapsible sets and the non-collapsible configurations. The final configuration set is plotted on the 2D graphs in Figure 4.2b. We can see that when the computation level is low (test-1), running all the modules on the mobile device (all-zero)

TABLE 4.1: Pareto efficient configurations of the Android-based MC hybrid applications obtained are listed. They are obtained as a result of offline profiling of the MC hybrid apps. The number of samples obtained of the configurations, and the efficiency of each configuration is stated. The configurations are of different granularities.

MC Hybrid App	**Configuration**	**Granularity Level**	**Samples**	**Efficiency of Configurations**				
				Runtime (sec)	**Battery Power Consumption (mJ)**		**Network Bandwidth Usage (KB)**	
				Mean	Mean	Standard Deviation	Mean	Standard Deviation
Proto App (Test-1)	00	class	100	2.99	279.645	70.7518	0	0
Proto App (Test-2)	00	class	100	3.02	614.6129	98.5792	0	0
	01:010	hybrid	100	3.00	538.8065	83.726	2.2568	0.0871
	10:001	hybrid	100	3.00	378.0	74.2911	3.8861	0.0836
	10:011	hybrid	100	3.01	344.4839	74.5848	5.6719	0.0997
Proto App (Test-3)	01:000	hybrid	100	9.35	5241.2903	1162.5894	0	0
	01:010	hybrid	100	5.99	2204.6774	510.2685	2.3155	0.0682
	1010	method	100	5.01	958.9355	158.1123	4.4542	0.0683
	01:110	hybrid	100	2.99	504.2258	89.6743	5.84	0.0803
ImageEffects	1010:00000000	hybrid	30	10.03	2666.685	587.5	133.0	0.8
	1000:010000	hybrid	30	8.13	1166.69	267.0	669.7	1.3
	1011:110000100	hybrid	30	8.02	1066.7	308.9	1065.7	1.4
	1000100000	method	30	8.09	967.569	279.3	1201.3	1.5
Mather	01:0000	hybrid	100	20.06	16756.871	3096.783	0	0
	001000	method	100	20.12	14737.9677	2970.083	3.072	0.026
	10:0110	hybrid	100	20.07	10895.452	2754.441	3.728	0.047

with class-level configuration is the right choice. In other words, there is no need for using code-offloading or executing the app with fine or hybrid-level granularity configurations when the app is not doing too much computation. However, when the computation level increases (test-1 and test-2), we get a Pareto-front (shown with a red line). The configurations forming the Pareto-front are the Pareto-optimal configurations. These are efficient in terms of battery power consumption and network usage. The efficient configuration in Test-1 and the Pareto-optimal configurations of test-2 and test-3 of the Proto app are shown in Table 4.1. We can see a mixed level of granularities of the efficient configurations, which shows that multi-level granularity is important for achieving optimisation in HMC applications.

The mean execution time of the Proto application for the Pareto-optimal configurations is also stated as a runtime in Table 4.1. We can see that as the computation level increases (from test-1 to test-3), the execution time of the app with all-zero configurations increases. The execution time of the hybrid-level efficient configuration $(01:110)$ in test-3 is same as the class-level efficient configuration (00) in test-1. In other words, with a maximum computation level (test-3), offloading three out of four modules $(01:110)$ took the same time as running all the four modules (00) on the mobile device and with less computation level (test-1). This shows that the performance of an HMC application can be improved while using multi-level granularity for achieving optimisation in HMC applications.

Power and Network consumptions per modules running on the cloud

We have tracked the behaviour of the Proto application executing the same number of modules on the cloud with different configurations, as shown in Figure 4.3. The mean power consumption and network usage per number of modules offloaded are shown in Figures 4.3a and 4.3b respectively. We can see in Figure 4.3a, when the computation level was low (test-1), using code-offloading was *energy-inefficient*. The power consumption was increasing per number of modules using code-offloading. However, as the computation level was increasing (test-2 and test-3), we can see that the code-offloading was becoming *energy-efficient*. The high number of modules executing on the cloud was consuming less battery power of the mobile device. This implies that running an HMC application with all-zero configurations, when the computation level of the app is low, can result in less power consumption than running on all-one configurations. On the other hand, when the computation level of the app is high. Executing it with all-one configurations can result in less power consumption than running on all-zero configurations. Moreover, we can see in Figure 4.3a (during test-2) the power consumption using method-level configurations (fine-grained) is comparatively higher than the class-level and hybrid-level configurations. This phenomenon is due to the fact that the decision to offload a method is checked for every offloadable method in each class. On the other hand, for a class-level configuration there is only one checking point for the whole class.

The network usage of the Proto app, running the different number of modules on the cloud, for mixed granularity level of configurations is shown in the line graph in Figure 4.3b. The network used by the modules of the app using code-offloading was the same in the three tests. It is because the input parameter "x" of the methods was an integer value. Also, the result generated by the modules was an integer value. The small difference between the network usage, which can be seen in the graphs, is due to the network or platform overhead.

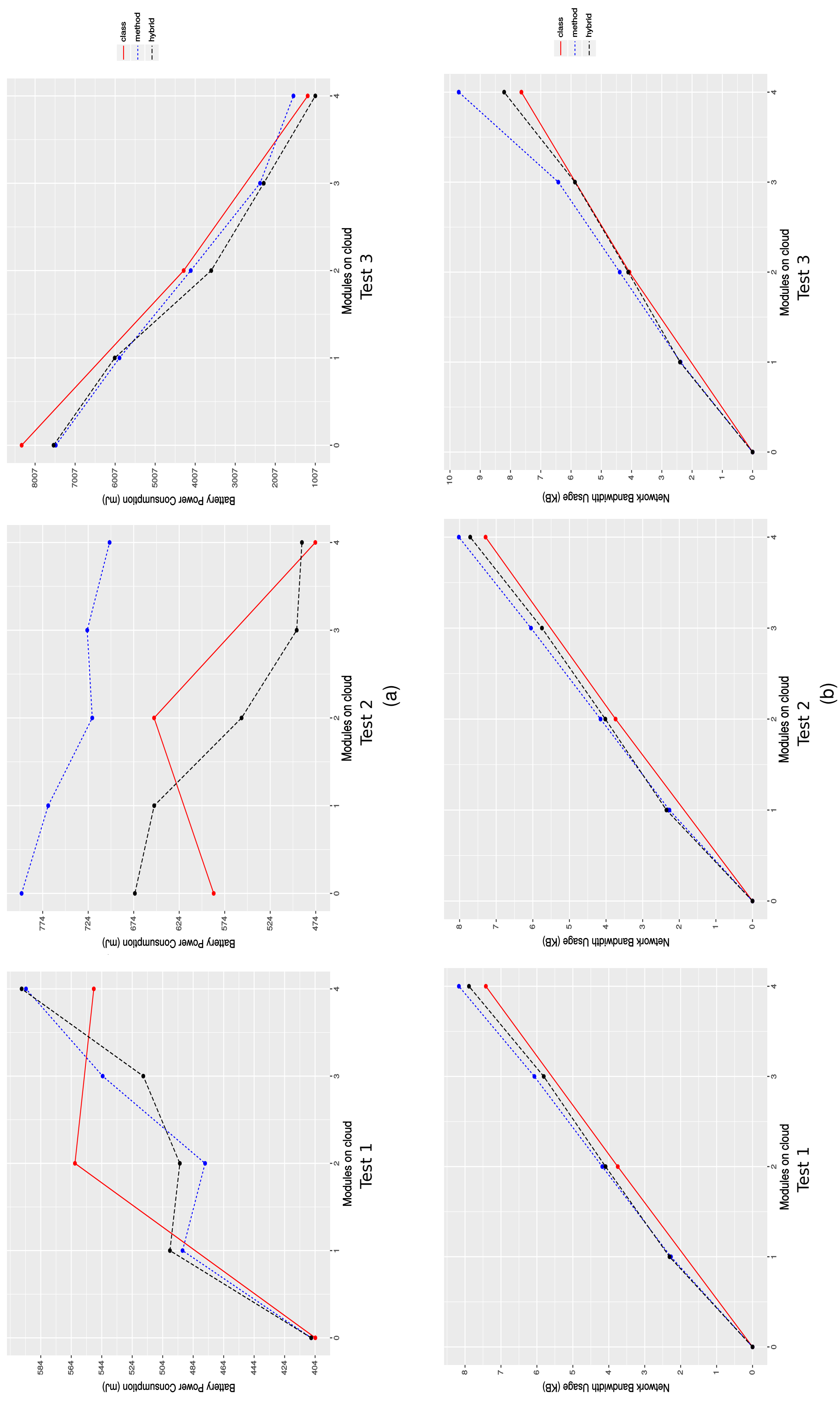


FIGURE 4.3: Plots showing the results obtained from three different profiling tests. The tests were carried out with a prototype Android-based application (Proto) to understand the underlying mobile-cloud application framework. The computation level of the app increases in each test subsequently. The plots in (a) and (b) shows the power consumption and network usage per number of modules running on the cloud respectively.

4.5.3 Results for the Android-based HMC Applications

In the previous subsection, we have discussed the behaviour of the MC application framework using a Proto app. Which gave us insight to better understand how the configurations of the two HMC applications (ImageEffects and Mather) will form after executing them using the MC application framework. The total number of configurations for ImageEffects were 3040. For Mather, we had a total of 83 applicable configurations. We were able to get 30 and 100 samples of all the configurations of ImageEffect and Mather respectively. In the below subsections we discuss the results for each of the apps separately.

Results for ImageEffects

The mean of power consumption and network usage of the configurations of ImageEffects are plotted on a 2D graph and shown in Figure 4.4a. We can see the configurations forming into two vertical-type clusters, which is similar to the behaviour of clusters of configurations, discussed in Section 4.5.2. The configurations that consumed more battery power are in the right-side cluster. The majority of these configurations are those that execute the modules involved in the image processing task on the mobile device. We can also see that the all-zero configurations, which use no code-offloading, are at the bottom of the graph. These configurations used a small amount of network - only for uploading/downloading images to the image server. The all-one configurations are in the left side cluster, except the method-level (1111111111), as they remotely execute the modules and consumed less battery power. The all-one method-level configuration (1111111111) is near the top of the right side cluster. This is due to the flip-flop pattern of data send and receive over the network for the maximum time (in the case of 1111111111 is ten times), which use more battery power and network overhead. The flip-flop pattern in all-one class-level configuration (1111) is the lowest (four times). Therefore, we can see it uses less network than other all-one configurations.

The filtered configurations in the collapsible sets along with non-collapsible configurations are shown in Figure 4.4b. We can see the Pareto-optimal configurations forming the Pareto-front. They are also listed in Table 4.1. These configurations are efficient in terms of minimum network usage and battery power consumption. Also, we can see that they are of different granularity level (method-level and hybrid-level), which shows that multi-level granularity is important for optimisation of HMC applications.

Figures 4.4c and 4.4d show the power and consumption of multiple granular configurations. We can see that running a maximum number of modules on the cloud consumed less power and network for class and hybrid-level configurations. The method-level configurations, due to the flip-flop pattern, consumed more data and power when running maximum modules on the cloud. However, they are efficient in terms of battery power consumption when running between one and five modules on the cloud.

Results for Mather

Mather evaluates mathematical expressions as discussed in Section 3.4.1. For the experimental purpose, we evaluate a $10 * 10$ matrix multiplication. The results obtained are plotted on graphs in Figure 4.5, where the all-zero and all-one configurations are also labelled. As in the Proto app and ImageEffects, the all-zero configurations of Mather are also at the bottom as they do not use the network for code-offloading. The filtered configurations in the collapsible sets along with the non-collapsible configurations are shown in Figure 4.5b. We can see the Pareto-optimal configurations form the Pareto-front and these configurations are also stated in Table 4.1. They are of the method and hybrid-level of granularities. The resultant shape of the Pareto front for Mather is a concave shape. While this is not typically expected for min-min problems, it is possible here, since;

1. For Mather only 75% of the configurations were feasible, which means that the search-space was restricted.

2. The two objective functions are not independent. This dependency is due to the fact that the transmitter also uses power when sending or receiving data. So when a configuration sends/receives more data, it also uses more power and will be shifted towards the upper-right of the graph. This can result in a concave Pareto front.

The Figures 4.5c and 4.5d shows the power consumption and network usage by number of modules offloaded to cloud with multiple granular configurations respectively. Due to the fact that all the configurations in search space for Mather were not applicable, the power consumption forming a saw-tooth effect per number of modules on cloud.

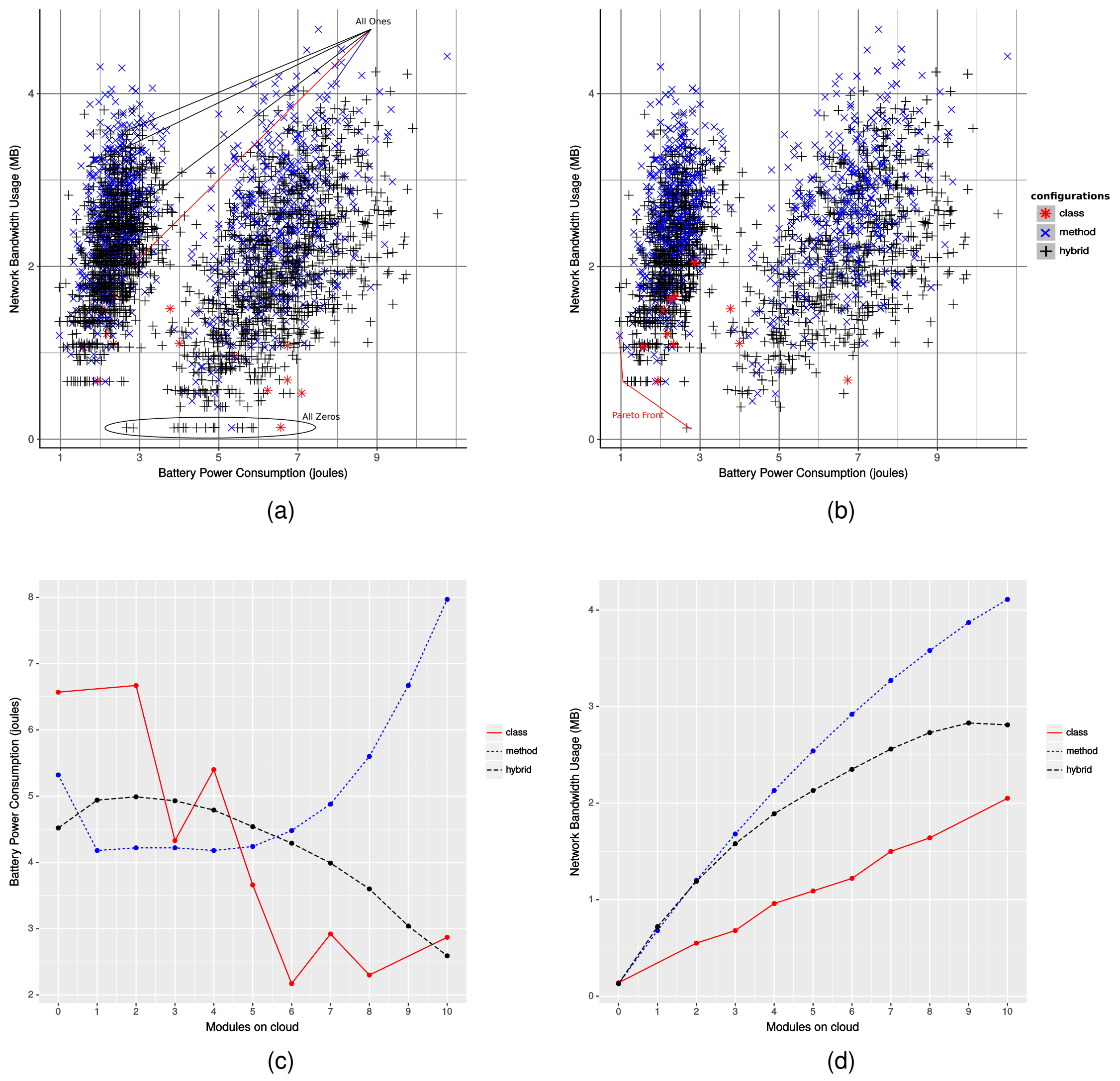
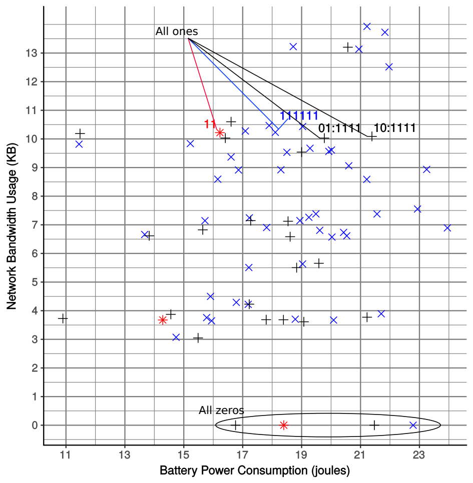


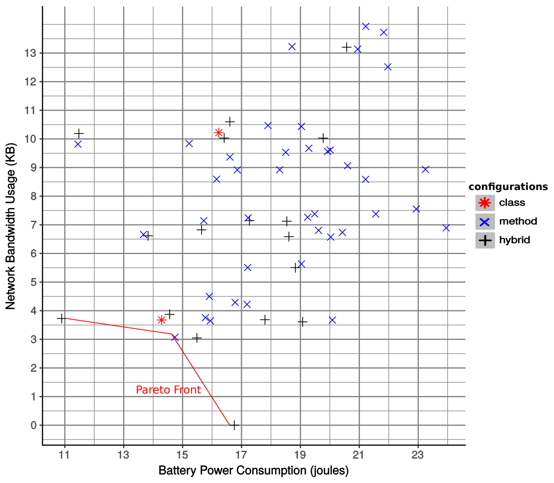
FIGURE 4.4: Plots showing configurations for ImageEffects HMC application. (a) All configurations of mixed granularity forming in two vertical shape clusters. The configurations running the computationally-intensive modules of the app are in the right side cluster. (b) Filtered configurations in the collapsible sets along with the non-collapsible configurations. The Pareto efficient configurations making the Pareto front are also shown. (c) Battery power consumption and (d) network usage per number of modules running on the cloud. Each line represent different granularity level of the configurations running the modules.

4.5.4 Results for Robotic Task

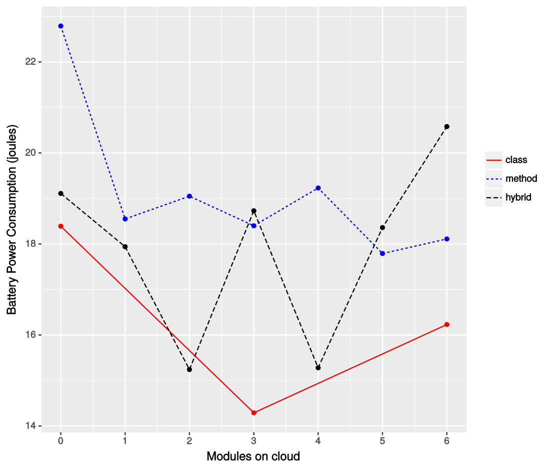
For the HMC foraging task, performed by Thymio robot, we did preliminary offline profiling first. This includes taking 30 samples of all of its class-level configurations and 1 sample of all of the method and hybrid-level configurations. As the total number of method-level and hybrid-level configurations were 16, 384 and 16, 000 respectively, it is not feasible to exhaustively search them in a limited amount of time. From the one sample of method and hybrid-level configurations (taken during preliminary profiling), we then select the best 150 configurations on the bases of minimum battery power and network usage. We took 29 more samples of the selected configurations, resulting in a total of 30 samples. These configurations (all of class-level, selected 150 method-level and hybrid-level) are plotted in Figure 4.6a. We can see the all-zero configurations of



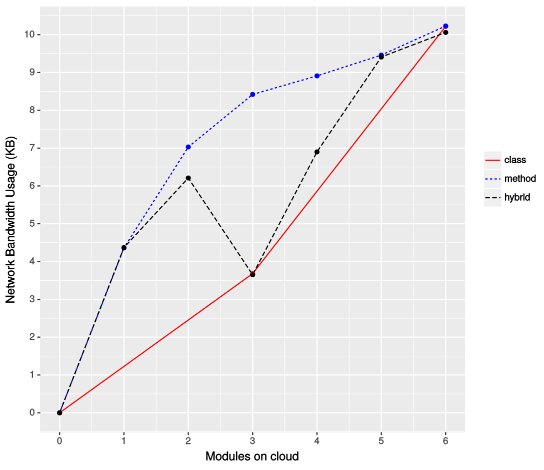
(a) All feasible configurations of mixed granularity for Mather application.



(b) Filtered configurations in the collapsible sets along with the non-collapsible configurations. The Pareto front is labeled.



(c) Battery power consumption per number of modules running on the cloud.



(d) Network usage per number of modules running on the cloud.

FIGURE 4.5: The results obtained from offline profiling of the Mather (an HMC application) are plotted in these graphs.

mixed granularity at the bottom of the plot, as they are not using the network due to code-offloading.

To achieve an efficient HMC foraging task, we applied the filter to these configurations (discussed in Section 4.4). We divided these configurations into collapsible sets, in which we only picked the non-dominated configuration(s) along with their statistically-significant configuration(s) if they were present. In the end, we created a final set of configurations by combining all the collapsible sets along with the configurations which were not collapsible. The configurations in the final set are plotted in Figure 4.6b, which also shows the Pareto front. The non-dominated configurations in the final set are the

TABLE 4.2: Offline profiling of the mobile-cloud foraging task carried out by a Raspberry Pi controlled Thymio robot. The number of samples obtained of the configurations and their mean and standard deviation of battery power consumption and network bandwidth usage along with total samples and their average execution time are stated.

Configuration	Granularity Level	Samples	Runtime (secs)	Battery Power Consumption (Joules)		Network Bandwidth Usage (kB)	
			Mean	Mean	Standard Deviation	Mean	Standard Deviation
1011:001000000001	Hybrid	30	32.6	0.2849	0.0113	38.2503	0.0695
00100000000000	Method	30	32.6	0.2958	0.018	15.9743	0.0971
00000000000000	Method	30	32.6	0.3061	0.0136	0	0

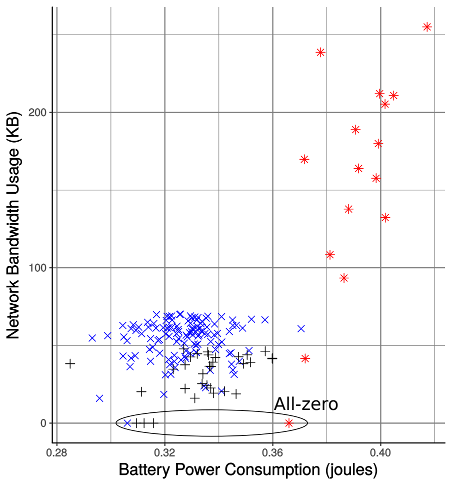
Pareto-optimal configurations, listed in Table 4.2. These configurations are superior to the others when the objectives are considered [117]. Also, these configurations optimize the efficiency trade-off and provide efficient alternatives to the HMC application in terms of battery power consumption and network usage. As offline profiling was carried out in a place near to the wireless base station with no interference, the mean runtime of all the configurations was the same.

The Figures 4.6c and 4.6d shows the power consumption and network usage by some modules offloaded to the cloud with multiple granular configurations respectively. We can see that the method-level and hybrid-level modules on the cloud are not complete for the full number of modules. It is because we select 150 number of method-level and hybrid-level configurations in the preliminary profiling, in terms of minimum power and network usage.

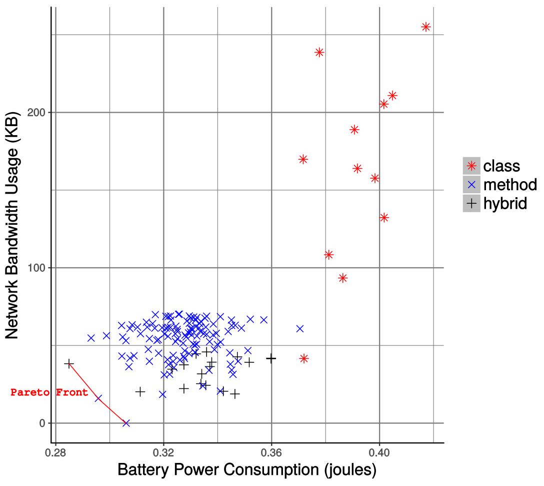
4.6 Limitations of the Approach

In this chapter, we have discussed a workflow to obtain energy efficient configurations of HMC applications. The following are some of the limitations.

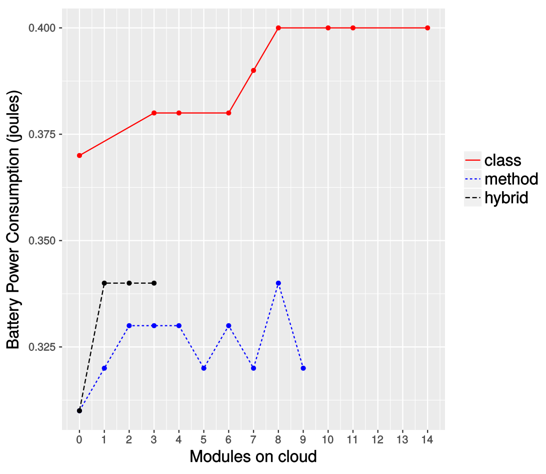
1. The workflow is using HMC application framework that are build for Android-based and Linux-based applications. Other mobile OSs such as IOS for iPhones or



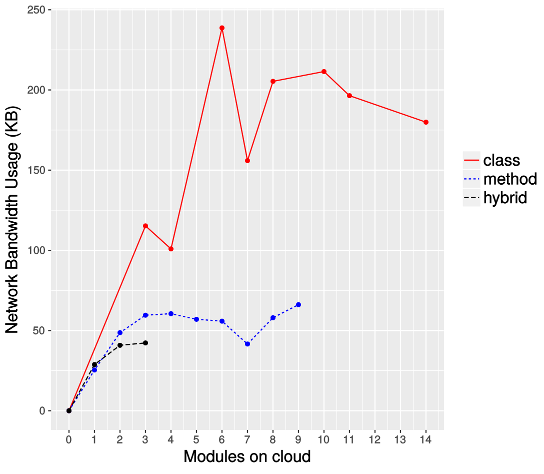
(a) Configurations of mixed granularity for the foraging task.



(b) Filtered configurations in the collapsible sets along with the non-collapsible configurations. The Pareto front is labeled.



(c) Battery power consumption per number of modules running on the cloud.



(d) Network usage per number of modules running on the cloud.

FIGURE 4.6: The results obtained from offline profiling of the HMC foraging task, performed by a battery powered and Raspberry Pi controlled Thymio robot, are plotted in these graphs.

iPads are not supported by the framework.

2. The offline framework discussed in this chapter, we have used exhaustive search algorithm to find the Pareto-optimal configurations. For HMC applications that have more than 15 offloadable modules, the exhaustive search algorithm will be time consuming to use in the offline profiling.

3. The HMC application framework works only for applications using Object Oriented Programming paradigm. Other paradigms such as functional programming and procedural programming are not supported.

4.7 Summary of Contribution

In this chapter, we discussed a workflow that can be used to achieve energy-efficient hybrid mobile-cloud applications. The workflow uses multi-objective optimisation to find a balance between the two conflicting objectives we consider: minimising battery power consumption and minimising network usage. We discussed how the objective functions are measured to obtain the efficiency of configurations. The statistical tests are done to filter statistically significant and non-dominated configurations in which the Pareto-optimal configurations are obtained. The Pareto-optimal configurations optimise the efficiency trade-off between power consumption and network usage.

We applied the workflow on two Android-based HMC applications (ImageEffects and Mather) and a foraging task performed by a battery-powered and Raspberry Pi controlled Thymio robot. Using offline profiling the Android-based applications and the robotic task was instrumented to find the efficiency of their configurations. Python-based scripts that implement exhaustive search algorithms were used to automatically and repeatedly executes the applications and the task. To understand the behaviour of the MC framework and to use the workflow to optimise the efficiency trade-off, we evaluate the obtained results with a Proto application we created. From the results analysis, we highlight the following main contributions.

1. Using the hybrid mobile-cloud framework (discussed in Section 3.3) in line with the workflow, can achieve energy-efficient mobile computing systems (smartphones

and robots). The efficiency trade-off between power consumption and network usage was optimised for Android-based HMC applications and Python-based tasks performed by mobile robots.

2. Computation offloading is energy effective when computationally-intensive modules are offloaded to the cloud. The total power consumption, including computing and communication power, is low in this case.

3. Computation offloading is in-effective when less computationally-intensive modules are offloaded to the cloud. In this case, the total power consumption is lower when they are executed on mobile devices.

4. The mobile applications that have low computation level will result in the formation of horizontal type clusters of configurations that execute the same number of modules on the cloud when code offloading is used. On the other hand, mobile applications having high computation level will result in the formation of vertical type clusters of configurations.

5. The efficient configurations obtained for the HMC applications had mixed granularity levels. This shows that the granularity level is important to consider during code offloading, to achieve energy-efficient hybrid mobile-cloud applications.

As mentioned in Section 4.2, the exhaustive search algorithm was used during offline profiling. The time it takes to exhaustively search the configurations depends on the total number of modules. Therefore, the larger the configuration set the more time the exhaustive search will take to find the efficient configurations. Estimated time for more than 20 modules would be more than a month for one execution for all configurations. Also, the exhaustive search algorithm searches for every configuration of any level of granularity, which takes a long time. Therefore, evolutionary algorithms such as Genetic Algorithms (GAs) and a more intelligent search algorithm can be implemented. The GAs take less time to converge, albeit to approximate optimal solutions. They will be discussed in the next chapter.

Chapter 5

Scalable Mobile-Cloud Hybrid Computing Systems

In this chapter, we will discuss search algorithms that could be used to find approximate to Pareto-optimal solutions and provide scalable options for HMC applications in terms of their size proportion to their offloadable modules. For a large number of offloadable modules, using the exhaustive search algorithm (to find Pareto-optimal configurations) is not a feasible option, regarding the time it takes to search the configuration sets. Therefore, depending on the time constraint, it may be suitable to get an approximation to the Pareto-optimal configurations in a reasonable amount of time. Multi-objective optimisation (MOO) algorithms represent a viable alternative to find this Pareto-optimal approximation set in one run potentially. We will use two multi-objective optimisation algorithms, which will be applied in this work: NSGA-II [42] and Two-Step search algorithm. We will compare the performance of these algorithms in terms of the quality of the algorithms' outcomes.

5.1 Challenges in Scaling Up

In order to find the Pareto-optimal configurations, we previously have used the exhaustive search algorithm in the offline profiling discussed in Section 4.2. The problem is that the exhaustive search algorithms enumerate all configurations present in the set in order to determine the energy-efficient configuration(s). If the total number of configurations is large, the time to complete the exhaustive search will be high. To give an estimation:

- For a total of 83 configurations of Mather (discussed in Section 3.4.1), the offline profiling took around two weeks to run the exhaustive search algorithm 30 times independently.
- For a total of $3,600$ configurations of ImageEffects (discussed in Section 3.4.1), the offline profiling took more than a month to run the exhaustive search algorithm 30 times independently.
- For a total of $32,400$ configurations of the HMC foraging task (discussed in Section 3.4.2), the offline profiling took more around a month for only one run of the exhaustive search algorithm.

TABLE 5.1: The number of configurations increases exponentially when the offloadable modules increases.

Modules	Configurations
8	256
14	16,384
20	1,048,576 million

The total number of possible configurations increases exponentially (2^n) when the number of offloadable modules, n, are increased for an HMC application. As stated in Table 5.1, it would take too much time to search efficient configurations for 20 offloadable modules. Such problems for which no known algorithm can find a solution to exact optimum in feasible amount of time are called NP-hard [56]. Therefore, to find Pareto-optimal configurations for the HMC applications performing robotic tasks, exhaustive search algorithms are best to use for only a small configuration sets.

5.2 Approaches to Scale Up

The term "scaling up" is used in different ways. Broadly, it refers to "doing more". For example, adding more resources to an existing system to reach a desired state of performance. In the case of our workflow, scaling up is required so that an approximation to the Pareto-optimal configurations by a search algorithm, in a reasonable amount of time, can be achieved. To potentially find this Pareto-optimal approximation set in one run, we discuss in the following subsections two search algorithms.

5.2.1 Two-Step Search

The applications designed for mobile devices (smartphones, robots) use the available device-limited libraries or resources such as GPS, sensors. Some classes that use them are not fit for offloading to the cloud as a whole. This results in very few numbers of classes that are fit for offloading either fully or partly. On the other hand, the methods in offloadable classes are usually in a high number. Therefore, the class-level configuration set in HMC applications, comparatively smaller than the method and hybrid-level, can be searched exhaustively in feasible time.

We design the Two-Step search algorithm to explore the configuration sets of HMC applications in two searching steps: 1) The first step aims to explore the class-level configuration set exhaustively to obtain the best class-level configurations. 2) The second step aims first to search the collapsible method and hybrid-level configurations of the obtained class-level configurations in step 1 and then randomly search their neighbour configurations.

The class-level configurations obtained in step 1 are the Pareto-optimal. We create their collapsible method-level and hybrid-level configurations. The neighbour configurations are obtained by flipping the bits of the collapsible configurations randomly. Figure 5.1 illustrates how the Two-Step search algorithm works. The circles represent the limit of the neighbour search space.

Algorithm 2 presents the pseudocode for the Two-Step search. It starts by generating the class-level configurations, taking the number of class-level offloadable modules. In the first step, the efficiency of the generated class-level population is measured using offline profiling (in line 2). It finds the power and network usage by running HMC applications with the configurations. After the profiling is finished, it calculates fronts based on minimum power and network values of the configurations. The non-dominated solutions represent the Pareto-optimal approximation set.

For selecting the the best configurations from the class-level population, we apply the *crowding distance* [42] of NSGA-II to rank the configurations (in line 4). Crowding distance is widely used in MOO to measure the density of solutions surrounding a particular solution in an already sorted population according to each objective function. Alternatively, clustering of solutions using Euclidean distance between solutions can be

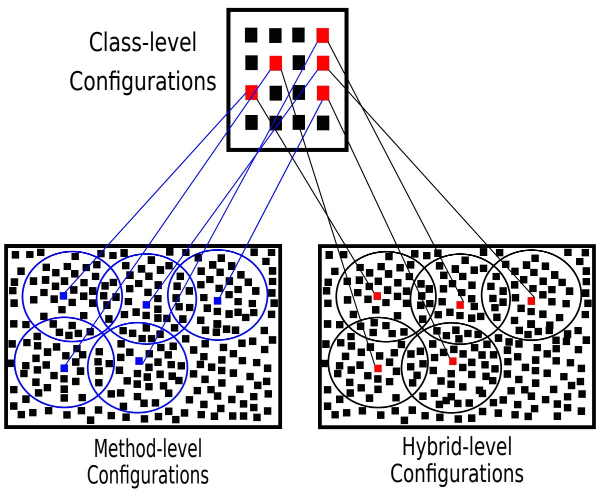


FIGURE 5.1: Two-Step search algorithm. The class, method and hybrid-level configuration sets are shown. The red dots in the class-level set represent the best configurations searched in the first step. The blue dots in method-level and red dots in hybrid-level configuration sets are the collapsible configurations of the respective best class-level configurations. The circles represent the neighbour configurations of the collapsible best configurations that can be searched.

used, which would require a very large amount of computation. Using the crowding distance metric, the computation is done in a fast manner. The selection starts with the best non-dominated front and iterates through all fronts. A configuration presenting a high crowding distance value is selected (in line 5). We select the best 5 configurations from the population as the *elitist* class-level configurations. In general elite means choosing the best of anything considered collectively. The selected elitist configurations are the best in the population considering the power and network usage.

The second step of the Two-Step search algorithm starts with finding the method and hybrid-level configurations (lines 6 and 7) of the elite class-level configurations of the first step. They are then used to execute the HMC application, which is profiling the app using these configurations (lines 8 and 9).

The Two-Step algorithm is used for the experimental work explained in the case studies section later in this chapter. In the experimental work, we will be expecting an approximation to the Pareto front we had in exhaustive search algorithm.

Algorithm 2 Two-Step Search Algorithm

```
1: $pop \leftarrow Population(MODULES)$                                ▷ First Step
2: PROFILE(pop, “classLevel")
3: pop.findfronts()
4: pop.findCrowdingdistance()
5: $elitePop \leftarrow pop.findClasslevelElitistPop()$
6: $methodLevelPop \leftarrow findMethodLevelPop(elitePop)$              ▷ Second Step
7: $hybridLevelPop \leftarrow findHybridLevelPop(elitePop)$
8: PROFILE(methodLevelPop, “methodLevel")
9: PROFILE(hybridLevelPop, “hybridLevel")
10: procedure PROFILE($pop, granularityLevel$)
11:     $robot \leftarrow connectToDevice()$
12:     for each $config \in pop.configs$ do
13:         $power, net, execTime \leftarrow getMeasurements(robot, config, granularityLevel)$
14:         pop.saveMeasurements(config, power, net, execTime)
15:     end for
16: end procedure
```

5.2.2 Evolutionary Algorithms (NSGA-II)

NSGA-II, a Non-dominated Sorting Genetic Algorithm II, is a fast elitist population-based algorithm for Multi-objective Optimisation [42]. It has the following features:

1. It uses an elitist principle, i.e., the elites of a population are given the opportunity to be carried to the next generation.

2. It uses an explicit diversity preserving mechanism (crowding distance).

3. It emphasises the non-dominated solutions.

In the Two-Step search algorithm, we have used NSGA-II in the second step. It starts by creating a population of individuals (i.e configurations). We used the binary representation of individuals as discussed in Chapter 3. As the representation does not carry the information of how many offloadable methods are in a class, we implemented this information in the algorithm. To select the parents in each generation, we used the tournament selection. To create a child, we used *Uniform Crossover* operator. With the crossover probability as 0.5, each gene (bit) is selected randomly from one of the corresponding genes of the parent chromosomes. We used *Flip Mutation* which works by flipping a gene (bit). With a very low mutation probability 0.2, it involves changing 0 to 1 and 1 to 0. We will discuss the genetic operators used in our implementation of NSGA-II in more details in the case study section of this chapter. For a comprehensive revision of the algorithm features, refer to [42]. Using offline profiling (discussed in Section 4.2), we implemented the NSGA-II algorithm in the Python script.

5.3 Case Studies

In the case studies, we will compare the outcomes of the two multi-objective optimisation algorithms (NSGA-II and Two-Step) discussed in Sections 5.2.2 and 5.2.1, respectively. Performance assessment includes both the quality of the outcome as well as the computational resources needed to generate this outcome. Concerning the latter aspect, the number of fitness evaluation will be the same for both algorithms. As to the quality aspect, comparing solutions in the presence of multiple criteria, the Pareto dominance concept must be used. However, when comparing two sets of solutions, some solutions in either set can be dominated by solutions in the other set, and some solutions can be incomparable.

As both algorithms contain randomness, due to the stochastic nature of the algorithms, for obtaining a well-based judgement related to the quality performance, it is necessary to perform any test over many algorithm runs. Therefore, to obtain meaningful statistical significant results, we executed both algorithms ten times, each time with a different seed for random number generator for both method and hybrid-level configuration sets.

To compare the quality of the solution sets produced by these two MOO algorithms, we used two different unary indicators: 1) hypervolume indicator (S-metric) [29] and 2) attainment surface [92]. These indicators are discussed in the following sub-sections.

5.3.1 Hypervolume Indicator

The hypervolume is a unary value, which is calculated as the sum of the areas formed by points on the non-dominated front and a chosen reference point (w). Figure 5.2a shows the area of the bi-dimensional region enclosed by a set of non-dominated points and a reference point (W) considering a minimisation problem. It is a well-known quality measure in evolutionary multi-objective optimisation to evaluate the performance of search algorithms [29, 24]. Hypervolume takes into account the diversity as well as the convergence of the non-dominated solutions. The reference point (W) represents some upper boundary of the region within all feasible points lie. The basic idea is that, for a bi-dimensional minimisation problem, the larger the area dominated by one non-dominated set in the objective space, the better the set is. To compare the performance of the two MOO algorithms, we will calculate the hypervolume of the obtained Pareto-optimal approximation set for each algorithm after each run.

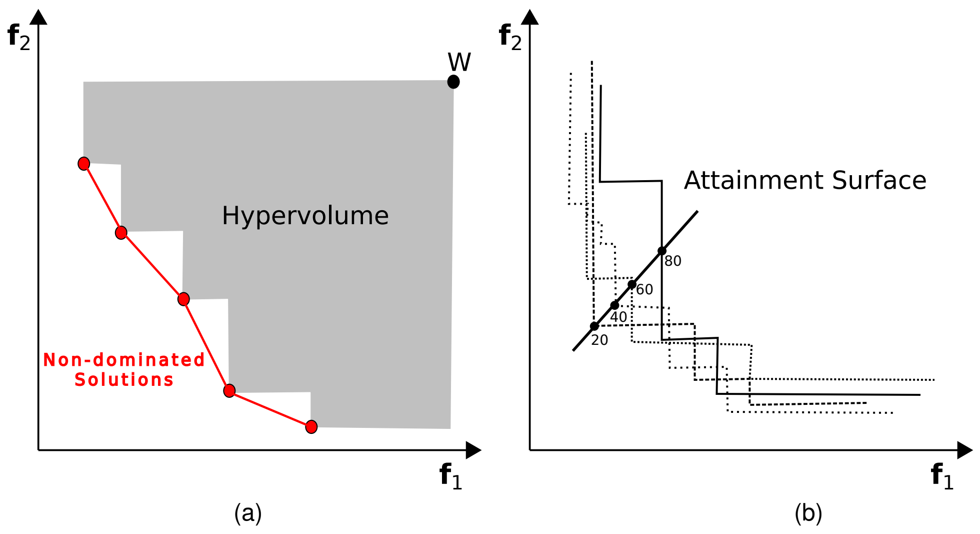


FIGURE 5.2: In (a) the hypervolume for a minimisation problem, calculated as the area enclosed by the non-dominated solutions and a chosen reference point (w), is shown. It computes the size of the region that the non-dominated points dominates. In (b) the attainment surface is created for a number of non-dominated solutions for a minimisation problem. It provides a description of the distribution of the obtained non-dominated set using the notion of goal-attainment.

5.3.2 Attainment Surface

The attainment surface corresponds to a region in the objective space which is attained by (dominated by or equal to) the good solution(s) returned by a MOO algorithm. It is formalised in the concept of the k%-attainment surface. Figure 5.2b shows the results obtained from arbitrary multiple runs of a MOO algorithm describing the distribution of the obtained non-dominated set using the notion of goal-attainment. The best attainment surface is the limit between the region attained by at least one run and the objective vectors never attained by any run. Whereas the worst attainment surface delimits the region attained by all runs. The graphical visualisation of attainment surface is a powerful tool providing a good insight into the algorithm performance. Lopez et al. [92] developed graphical tools for the analysis of bi-objective optimisation algorithms that plot the attainment surface of the solution sets. We will use the tool in R to plot the probabilistic distribution of the configurations along the Pareto front, obtained by the two algorithms. The tool calculates the empirical attainment function (EAF), which provides a summary of the outcomes of the ten different runs of each algorithm. By plotting and comparing the EAFs of the two algorithms, we will be able to pinpoint several performance behaviours.

5.3.3 Android Applications

The Android-based hybrid mobile-cloud applications (ImageEffects and Mather) were both used as test-bed applications for the exhaustive search algorithm. As discussed in Section 4.5, using the exhaustive search algorithm we were able to get the Pareto-optimal configurations for both these applications. As a rule of thumb [69], the objective functions evaluation was kept to 30 for each configuration using the exhaustive search algorithm.

In this case study, we will only consider the ImageEffects - an HMC Android-based application, because of its high number of method-level configurations $(1, 024)$ and hybrid-level configurations $(2, 560)$. For the Mather, the configuration set (having a total of 83 configurations) can easily be searched exhaustively to find the energy-efficient configurations.

For the NSGA-II implementation, we empirically chose some parameters based on results in the literature. The parameters were set as population size = 20, number of generations = 30 and the tournament population size = 10. The genetic operator parameters were set as crossover probability = 0.5 and mutation probability = 0.2. We executed both MOO algorithms (NSGA-II and Two-Step) ten times, and the number of objective function evaluation was kept the same (600) for both algorithms in each run. We compare the results of both algorithms with the result of the exhaustive search algorithm in terms of the hypervolume and attainment surface indicators in the following subsections.

Result of Hypervolume Indicator for ImageEffects

Figure 5.3 shows the mean hypervolume values of the method-level and the hybrid-level configurations, for the ten runs of NSGA-II throughout the generations. We can see that the algorithm has fully converged for both configuration sets.

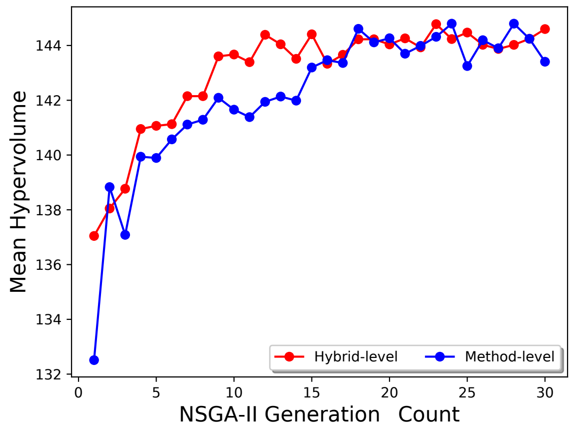


FIGURE 5.3: The mean hypervolume obtained for ten runs of NSGA-II throughout the generations. The dots represent the mean hypervolume level of the ten runs for the number of generations. The two lines are for the method-level and the hybrid-level configuration sets created for the ImageEffects.

To compare the performance of the two MOO algorithms and the exhaustive search algorithm in terms of the obtained non-dominated solutions, we have calculated the hypervolume in each run. For this comparison, we randomly chose ten runs of the exhaustive search algorithm from the results obtained previously (discussed in Section 4.5.3).

The box-and-whisker plot in Figure 5.4 shows the distribution of hypervolume values for the method-level configuration set. We can see that the exhaustive search algorithm finds near to true Pareto-optimal configurations, where the median of the hypervolume for ten runs is 0.98. There is variance in the obtained hypervolume values, where the standard deviation is 0.02. This is because of the noisy nature of the objective functions.

The medians of the hypervolume for both MOO algorithms are nearly at the same level (median for NSGA-II = 0.91 and median of Two-Step = 0.93). However, as we can see the underlying distributions are very distinct. It indicates a better consistency in the hypervolume values (for the ten runs) for the Two-Step algorithm when compared with NSGA-II.

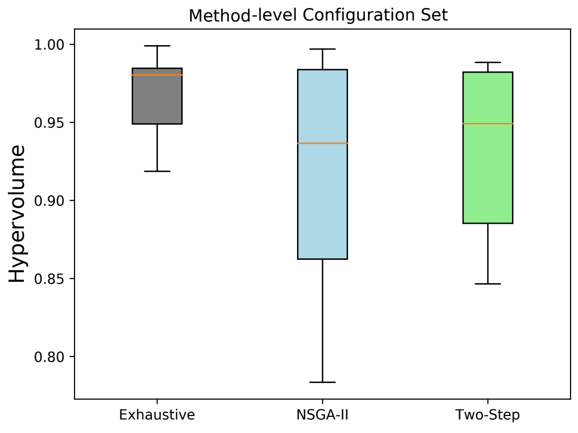


FIGURE 5.4: A box-and-whisker plot showing the hypervolume distribution of ten independent runs of Exhaustive search, NSGA-II and Two-Step algorithms. The non-dominated configurations obtained by the algorithms were for the method-level configuration sets created for the ImageEffects.

The box-and-whisker plot in Figure 5.5 shows the distribution of hypervolume values for the hybrid-level configuration set. Given the ten runs, the exhaustive search finds near to the Pareto-optimal configurations, as the resultant median is "0.99" with a standard deviation of "0.007.

The medians of the hypervolume for both MOO algorithms are nearly at the same level (median for NSGA-II = 0.984 and median of Two-Step = 0.989). However, as we can see the underlying distributions are very distinct. It indicates a better consistency in the hypervolume values (for the ten runs) for the NSGA-II algorithm when compared with Two-Step.

Result of Attainment Indicator for ImageEffects

To visualise the behaviour of the exhaustive search and the two MOO algorithms (NSGA-II and Two-Step), we plotted the attainment surface of the non-dominated solutions obtained by these algorithms. The attainment surface illustrates wherein the objectives space and by how much the outcomes differ for the method-level and hybrid-level configuration sets.

Figures 5.6a, 5.6b and 5.6c show the attainment surface of the obtained non-dominated method-level configurations by exhaustive, NSGA-II and Two-Step search algorithm respectively. Additional to the best and the worst attained surface, we have also shown

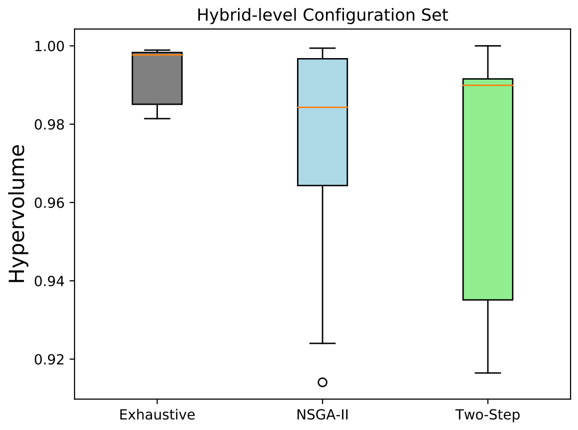


FIGURE 5.5: A box-and-whisker plot showing the hypervolume distribution of ten independent runs of Exhaustive search, NSGA-II and Two-Step algorithms. The non-dominated configurations obtained by the algorithms were for the hybrid-level configuration sets created for the ImageEffects.

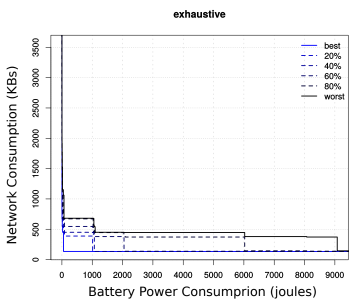
the attainment surface with other percentiles ($20\%, 40\%, 60\%, 80\%$). We can see that the attainment surface of method-level configurations obtained by the Two-Step search algorithm is more compact concerning all percentiles compare with NSGA-II algorithm.

Similarly, in the case of hybrid-level configurations, the attainment surface obtained by the Two-Step search algorithm is even more compact with respect to all percentiles as shown in Figure 5.7.

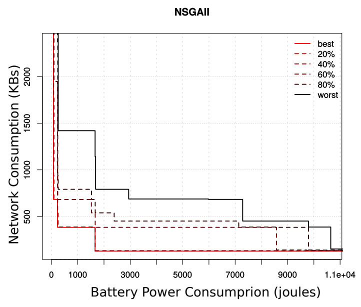
Figure 5.6d points out the differences between exhaustive and NSGA-II, while Figure 5.6e points out the differences between exhaustive and Two-Step algorithms; the difference is with respect to their corresponding EAFs. The value of the EAF indicates the probability of attaining an area in the objective space. The performance of an algorithm will be considered better than the other if its EAF value at a particular area is larger than the other. The grey level represents the magnitude of the difference. From the figures, we can see that in the two MOO algorithms, Two-Step search performs better in terms of finding good solutions towards minimisation of network usage in method-level configuration sets as well as in the hybrid-level configuration set as shown in Figures 5.7d and 5.7e.

5.3.4 Robotics

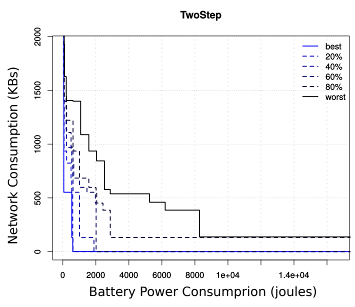
In this case study, we will consider the HMC foraging task performed by a battery-powered and Raspberry Pi controlled robot as a test-bet to compare the performance of the two MOO algorithms in terms of hypervolume and attainment surface indicators. We will use the method-level and hybrid-level configuration sets of the foraging task. The total number of configurations in the method-level set are $16,384$ and in the hybrid-level set are $16,000$. For the NSGA-II implementation, we empirically chose some parameters based on results in the literature. The parameters were set as population size = 10, number of generation = 30 and the tournament population size = 5. The genetic parameters were set as crossover probability = 0.5 and mutation probability = 0.2.



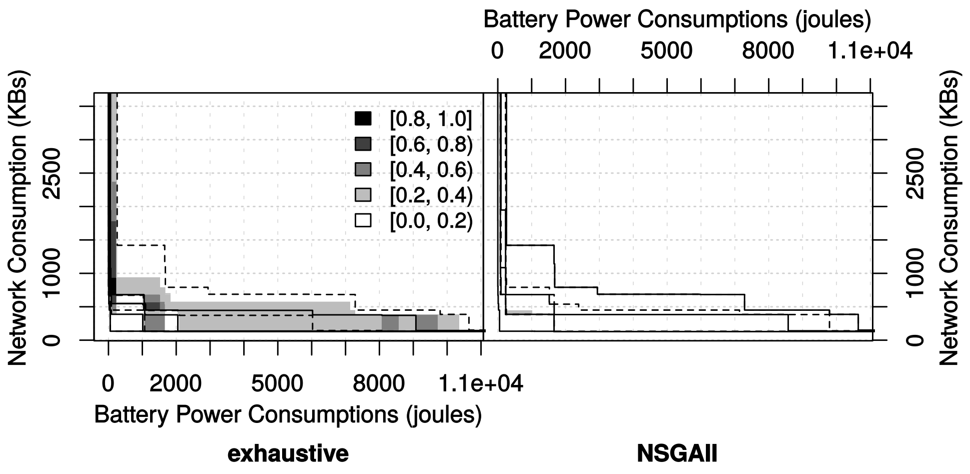
(a)



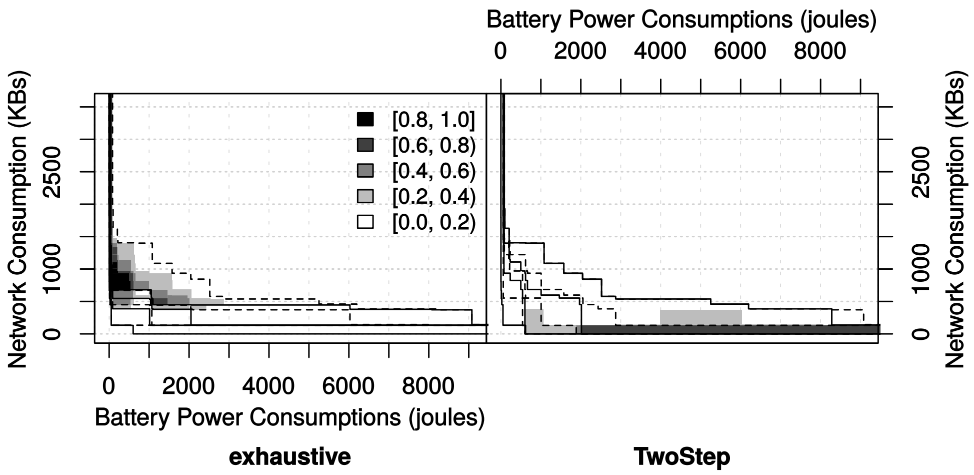
(b)



(c)



(d)



(e)

FIGURE 5.6: Ten independent outcomes obtained for the method-level configuration set of ImageEffects by the three different algorithms. With respect to six quartiles: In (a) the attainment surface of exhaustive search is shown, in (b) the attainment surface of NSGA-II is shown, and in (c) the attainment surface of Two-Step is shown. The location of the difference between the EAFs of exhaustive search and NSGA-II (d) and exhaustive search and Two-Step (e) is shown where the gray level represents the magnitude of the difference.

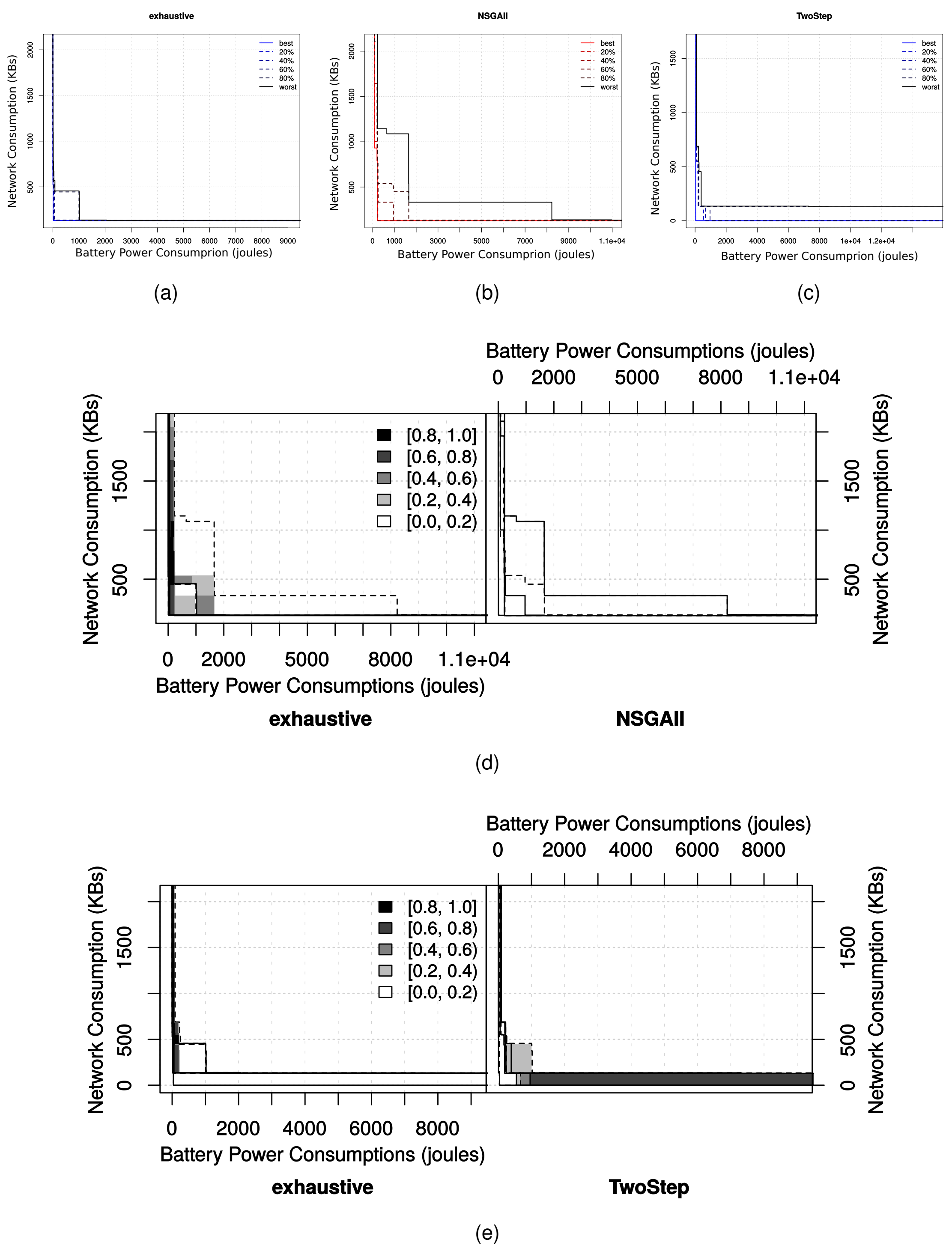


FIGURE 5.7: Ten independent outcomes obtained for the hybrid-level configuration set of ImageEffects by the three different algorithms. With respect to six quartiles: In (a) the attainment surface of exhaustive search is shown, in (b) the attainment surface of NSGA-II is shown, and in (c) the attainment surface of Two-Step is shown. The location of the difference between the EAFs of exhaustive search and NSGA-II (d) and exhaustive search and Two-Step (e) is shown where the gray level represents the magnitude of the difference.

Result of Hypervolume Indicator for the foraging task

We run each algorithm ten times, and the hypervolume was calculated for each obtained non-dominated solutions for both algorithms. The reference point used was the same in all obtained solutions in both algorithms. The box-and-whisker plots in Figure 5.8 and Figure 5.9 show the distribution of hypervolume values for the method-level and hybrid-level configuration sets, respectively.

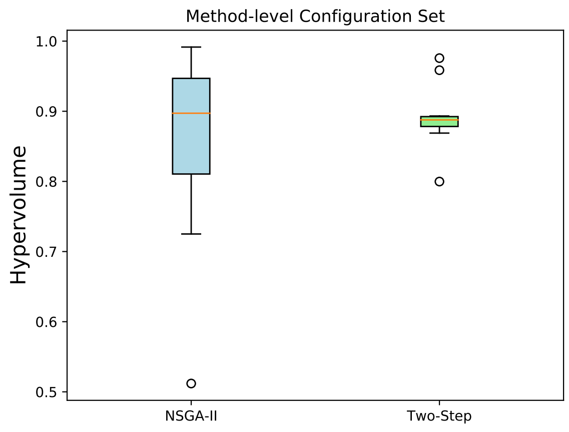


FIGURE 5.8: A box-and-whisker plot showing the hypervolume distribution of ten independent runs of NSGA-II and Two-Step MOO algorithms. The non-dominated configurations obtained by the algorithms were for the method-level configuration sets created for the foraging task.

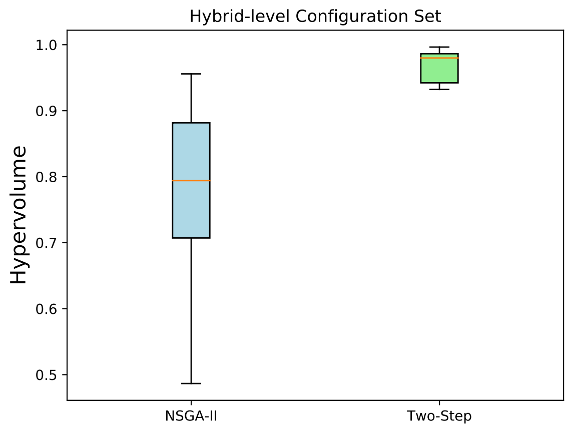


FIGURE 5.9: A box-and-whisker plot showing the hypervolume distribution of ten independent runs of NSGA-II and Two-Step MOO algorithms. The non-dominated configurations obtained by the algorithms were for the hybrid-level configuration sets created for the foraging task.

For the method-level configuration set, the medians of the hypervolume for both algorithms are nearly at the same level. However, the underlying distributions are very distinct. It indicates a better consistency in the hypervolume values (for the ten runs) for the Two-Step algorithm when compared with NSGA-II. Given the amount of time (two weeks) that both algorithms took to generate these results, we can say that the Two-Step algorithm performed well. If the time constraint is removed, the NSGA-II might perform better by increasing its number of generations. As we can see in Figure 5.10, the mean hypervolume value, corresponding to the mean hypervolume value for the ten runs throughout the generations, has not fully converged for the method-level configuration set.

For the hybrid-level configuration set, it is evident from Figure 5.9 that the Two-Step algorithm performed better than the NSGA-II. In the case of NSGA-II, the distribution of the hypervolume values shows a large spread. Similar to the method-level configuration set, the mean hypervolume value has not fully converged as shown in Figure 5.10.

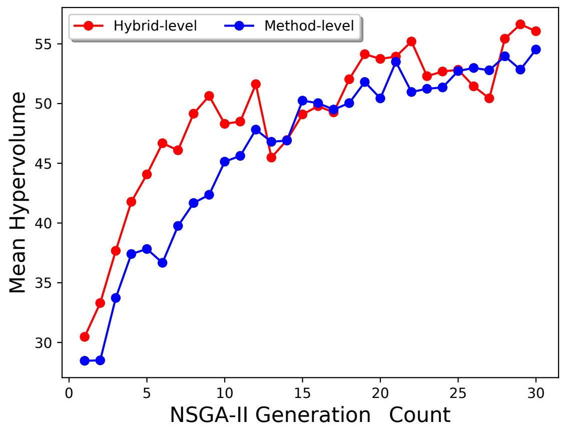


FIGURE 5.10: The mean hypervolume obtained for ten runs of NSGA-II throughout the generations. The dots represent the mean hypervolume level of the ten runs for the number of generations. The two lines are for method-level and hybrid-level configuration sets created for the foraging task.

Result of Attainment Indicator for the foraging task

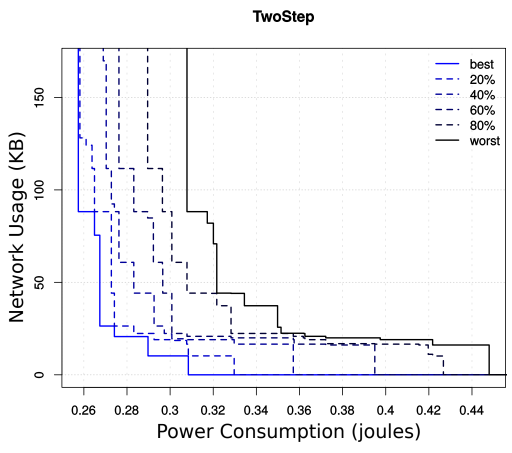
For the foraging task, to visualise the behaviour of the two MOO algorithms (NSGA-II and Two-Step) and illustrate where in the objectives space and by how much the outcomes differ for the method-level and hybrid-level configuration sets, we plotted the attainment surface of the non-dominated solutions obtained by the two algorithms.

Figures 5.11a and 5.11b show the attainment surface of the obtained non-dominated method-level configurations by NSGA-II and Two-Step search algorithm respectively. Additionally to the best and the worst attained surface, we have also shown the attainment surface with other percentiles $(20\%, 40\%, 60\%, 80\%)$. We can see that the attainment surface of method-level configurations obtained by the Two-Step search algorithm is more compact for all percentiles. Similarly, in the case of hybrid-level configurations, the attainment surface obtained by the Two-Step search algorithm is even more compact for all percentiles as shown in Figure 5.12.

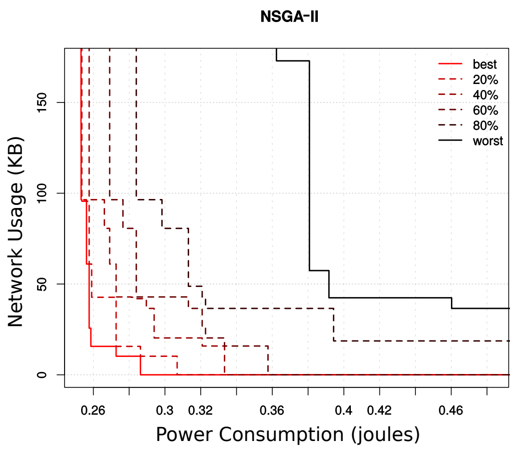
Figure 5.11c points out the differences between the two algorithms concerning their corresponding EAFs. The value of the EAF indicates the probability of attaining an area in the objective space. The performance of an algorithm will be considered better than the other if its EAF value at a particular area is larger than the other. The grey level represents the magnitude of the difference. From the figure, we can see that the Two-Step search algorithm performs better in terms of finding good solutions towards minimisation of network usage that was not good towards the minimisation of power usage. On the other hand, the NSGA-II performs better towards high-quality configurations for the minimisation of power consumption and also slightly for the minimisation of both objectives.

5.4 Summary of Contributions

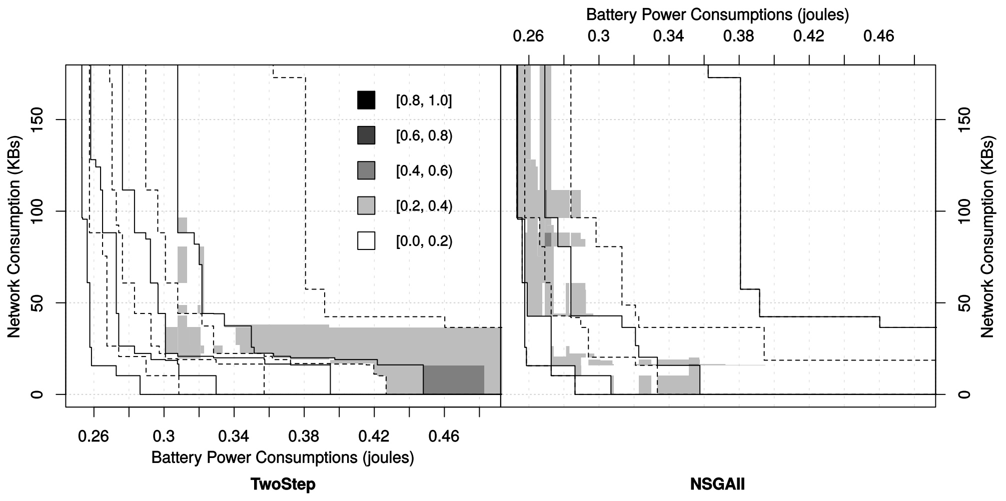
In this chapter, we discussed a solution to scale up the method of achieving energy-efficient HMC applications. The scale-up strategy is composed of using evolutionary algorithms, such as NSGA-II, and an intelligent Two-Step search algorithm; which we introduced for searching efficient configurations for HMC applications. The analysis of the obtained results provided some key facts regarding how much an HMC application for Android-based smartphones or a task for mobile robots can be scalable, proportional to the number of their offloadable modules, to produce efficient configurations that optimise the trade-off between power and network usage. Some key findings in line with the main contributions are enumerated as follow.



(a)



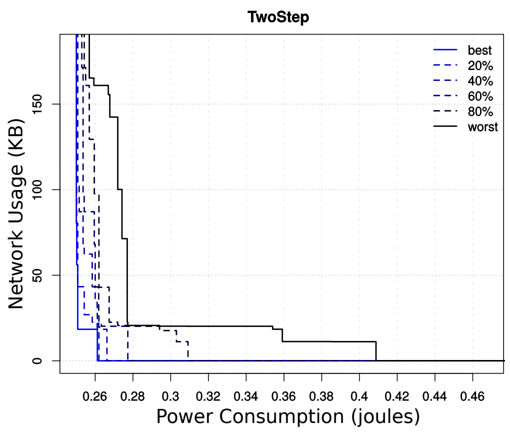
(b)



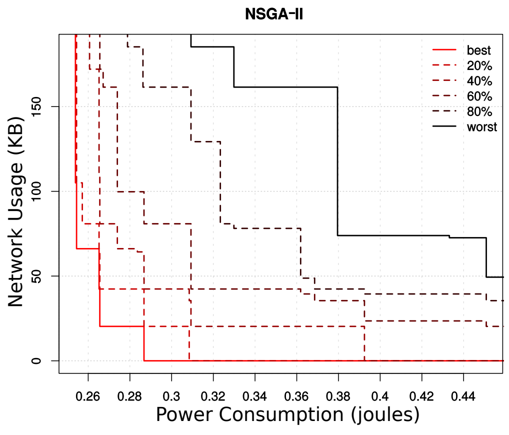
(c)

FIGURE 5.11: Ten independent outcomes obtained for the method-level configuration set by the two different MOO algorithms. In (a) the attainment surface of Two-Step is shown with respect to six quartiles. In (b) the attainment surface of NSGA-II is shown with respect to six quartiles. In (c) the location of the difference between the EAFs of the two algorithms is shown where the gray level represents the magnitude of the difference.

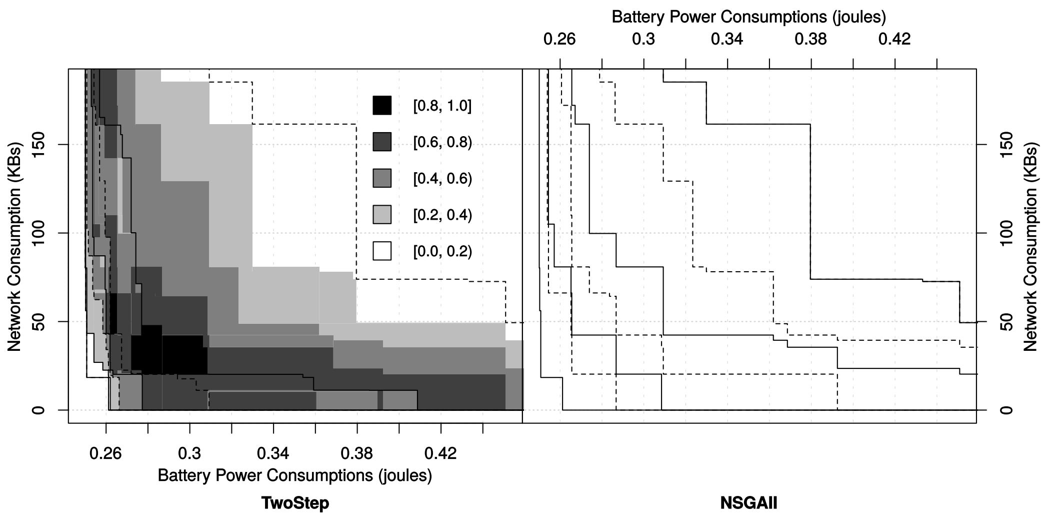
1. For small-scale HMC applications, the exhaustive search algorithm is appropriate to use for finding the Pareto-optimal configurations. In the Two-Step search algorithm, we used an exhaustive search to find Pareto-optimal class-level configurations (in the first step) because of a small number (4) of class-level offloadable modules for both ImageEffects and foraging task. The total number of configurations were 16 for which the exhaustive search algorithm took around 30 minutes to complete. However, in case of the offline profiling (discussed in Section 4.2), the



(a)



(b)



(c)

FIGURE 5.12: Ten independent outcomes obtained for the hybrid-level configuration set by the two different MOO algorithms. In (a) the attainment surface of Two-Step is shown with respect to six quartiles. In (b) the attainment surface of NSGA-II is shown with respect to six quartiles. In (c) the location of the difference between the EAFs of the two algorithms is shown where the gray level represents the magnitude of the difference.

exhaustive search took more than a month to find the Pareto-optimal configurations for the total number of configurations $(32, 400)$ in all the three sets (class-level, method-level and hybrid-level) combined of the foraging task. According to our experiments, offloadable modules between 2 to 8 would lie the so-called small-scale HMC applications. The class-level and method-level configurations of Mather are 2 and 6 respectively and can be considered a small-scale HMC application.

2. For medium-scale HMC applications, the exhaustive search algorithm will take a

significant amount of time to complete and, therefore, would not be practical to use. In this case, evolutionary algorithms, such as NSGA-II, are appropriate to use, which can approximate the Pareto-front solutions in a reasonable amount of time. For the ImageEffects and Foraging task, where the number of class-level configurations for both is equal to 16, and the combined method-level and hybrid-level configurations are equal to $3,584$ and $32,384$, both the Two-Step and NSGA-II were feasible to use. They took about one week for the ImageEffects and two week time for the foraging task, for the ten independent runs to complete. According to our experiments, offloadable class-level modules less than 8 and method-level modules higher than 8 would lie the so-called medium-scale HMC applications. ImageEffects and foraging task both can be considered medium scale HMC applications. For both of these two HMC applications, the Two-Step algorithm performed better than NSGA-II.

3. For large-scale HMC applications, the Two-Step search algorithm would not be practical to use as the exhaustive search in step 1 would not complete in a feasible amount of time. For such applications, finding a scalable solution is an open research question that we will leave for future work. According to our experiments, offloadable class-level and method-level modules higher than 8 would lie the so-called large-scale HMC applications.

Chapter 6

Self-Adaptive and Self-Aware Hybrid Mobile-Cloud Computing Systems

So far we have discussed achieving energy-efficient and scalable HMC applications developed for Android-based smartphones and Raspberry Pi controlled robots. They use code-offloading and multi-objective optimisation techniques to optimise the trade-off between battery power consumption and network usage. We created a workflow and instrument the applications to measure the efficiency trade-off, using offline profiling discussed in Section 4.2. The results of the offline profiling produced Pareto-efficient configurations. In this chapter, we will discuss how these configurations can be used together on-the-fly, while not compromising the performance of the HMC applications in terms of power consumption and execution time. We will discuss in particular about runtime decision mechanisms by employing self-adaptivity and self-awareness in our HMC framework. The runtime decision of the framework will be based on (1) changing of the environment (i.e. change in WiFi signal level with time), and (2) itself in a changing environment (i.e. actual observed packet loss in the network). Also, we will discuss a workflow that can be employed that can be used for online profiling.

6.1 Runtime Optimisation

In Chapter 4, we discussed how to instrument the HMC applications to measure the battery power and network usage during one complete run of the applications using offline profiling. In the case studies, discussed in Section 4.5, we carried out experiments to obtained Pareto efficient configurations of two Android applications and one robotic task. These configurations are energy-efficient and describe which modules of the applications should run on the device or the cloud. The offline profiling was carried out in a controlled lab environment where the mobile device was operated in the best coverage location of the wireless base station (i.e. WiFi router). As there were no obstacles in the middle that could obstruct the WiFi signals, there was no wireless interference during the communication between the device and the cloud. However, in real life, the external conditions change with the time and can affect the application's execution. For example, the mobile device moves to a location where it is: 1) subject to wireless interference, or 2) receiving good signals but there is a congestion in the network. In both scenarios, there will be packet loss during communication between the device and the cloud, which will result in network latency. The performance of applications will degrade when using code-offloading if there is latency in the network. By enabling the HMC framework with self-adaptivity and self-awareness makes it able to monitor its operative context to take run-time decisions, which is to use the right configuration and, therefore, avoid the network latency.

6.1.1 Self-Adaptive Hybrid Mobile-Cloud Computing (HMCC) Systems

Self-adaptivity can enable hybrid mobile-cloud applications to modify their behaviour at run-time, in response to the changing environment and make better decisions on how to use the available resources [100]. In the context of HMCC systems (i.e. smartphones and robots), as the mobile device moves, the WiFi signal level degrades over time. This is because of wireless interference caused by factors such as obstacles in the middle of the communication channel between a mobile device and the base station. The signal degradation can cause packet loss, which results in network latency. The packet loss in the network (if high) can cause long socket-wait time plus TCP retransmission at both endpoints (mobile and cloud).

We consider the WiFi signals as the changing environment, which changes across the network coverage area. In order to avoid network latency due to the low signal level, the MC framework is modified so that it can monitor the signal levels continuously and based on which makes self-adaptive decisions at runtime. If the signals are good, the framework will switch to a configuration that is using code-offloading. If the mobile device moves to an area where the receiving signal level is bad, then by using the self-adaptive decision mechanism the framework will switch to the all-zero configuration keeping the computation on the device. For example, if the signal level is degraded (low) enough to cause latency due to packet loss, the framework will switch to run on such a configuration that does not use the code-offloading.

Using the self-adaptivity in the proposed solution, the framework made decision based on the Wifi signal level changes. The signal level is monitored continuously, and when it is low enough and below a threshold the HMC application would switch to a configuration to execute all the modules on a mobile device. The signal level is monitored continuously by the application. Therefore, with the self-adaptive decision mechanism, battery power consumption of the device is minimised.

6.1.2 Self-Aware Hybrid Mobile-Cloud Computing (HMCC) Systems

In line with the definitions from Lewis et al. [85, 86] and Kounev et al. [81], we consider an HMCC system to be self-aware when it gathers knowledge, not just about the environment, but about itself in that environment, on an ongoing basis. It is then able to use this knowledge to drive its decision making at runtime. In a mobile-cloud (MC) scenario, the availability of a high-quality network connection is one of the key requirements for the mobile device to make effective use of code offloading [125]. While a self-adaptive system observing the environment may base decisions on environmental factors such as signal strength (as discussed above), self-awareness instead allows offloading decisions to be based on monitoring the device's behaviour within that environment, specifically in this case, its success in communicating over the network. We operationalise this here by enabling the device to monitor the level of packet loss while running the code-offloading.

When this is low, the self-aware MC framework offloads code. When the packet loss is sufficiently high, then the code is run on the device. By observing the actual runtime impact of attempting to run the offloaded code, the device is no longer required to use estimates, based on externally observable proxy features (i.e. signal strength).

6.2 Online Profiling

We use online profiling of the HMC applications (introduced n Section 3.4) to measure the power consumption, network usage and execution time while the external environment is changing with time. The difference between offline and online profiling is that in offline profiling we instrument the the HMC applications to find the efficient configurations. In the online profiling, the environmental keeps changes during the runtime and factors such as network delay and congestion add latency and effect the code-offloading of HMC applications. Online profiling aims to validate the self-adaptive and self-aware decision mechanism of our framework. For online profiling, we have created a controlled lab environment which can be used to instrument applications and keep changing the external operating environment. The lab environment is discussed as follow.

6.2.1 Experimental Setup

We created a lab-based controlled setup for online profiling. While a mobile device remains stationary during the online profiling, we used a bunch of tools to simulate the packet loss on an intermediate node between the mobile device and the cloud server in a wireless network.

As shown in Figure 6.1, a mobile device such as an Android smartphone or a Raspberry Pi controlled robot can be connected to a PC via a USB cable of accessed via SSH. We wrote Python-based scripts that automates the workflow for online profiling. The workflow is to execute the HMC applications on the mobile device and measure the power consumption, network usage and execution time during the runtime. During the code-offloading, the data packets between a mobile device and the cloud server are passed through the PC (an intermediate node), which controls the flow of the packets. This is to simulate the *latency* in the wireless network. The offline profiling is performed for static offloading, no offloading, self-adaptive and self-aware decision mechanisms of the framework.

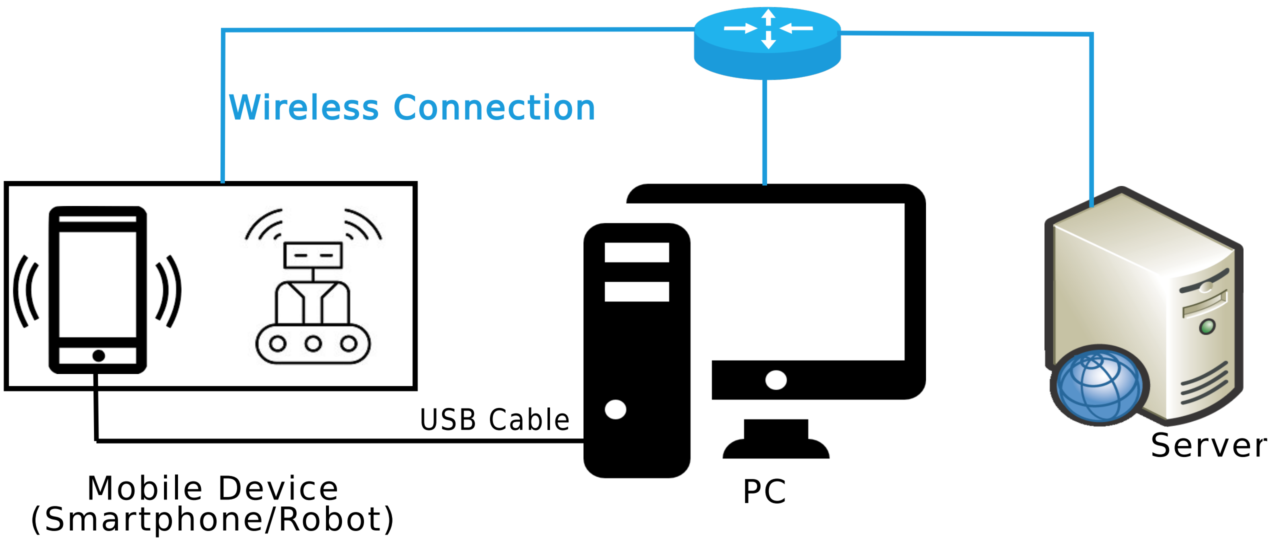


FIGURE 6.1: A mobile device (an Android smartphone or a Raspberry Pi controlled robot) connected to a PC. Scripts running on the PC execute the HMC applications on the mobile device and controls the flow of packets between the mobile device and the server.

6.2.2 Network Latency

The delay of data packets in a network, incurred during communication of a message, from a source node to its target node is called latency. It is usually measured as a round trip delay, which is the time taken for data to reach the target node and come back again to the source node. An HMC application, using code-offloading, sends a limited amount of data to the cloud (using TCP/IP network) and then wait to receive an acknowledgement before sending more data. If there is latency in the network, the MC application will experience long socket-wait time and will retransmit the TCP packets with the cost of using more battery power and network usage. Following are the two broad categories that can cause latency in the network.

Wireless Interference

Wireless interference causes signal level to degrade and typically comes from factors such as physical barriers, i.e. concrete, metal, mirror, wood and water. Other factors like frequency interference can also cause the signals to degrade. This happens when another signal crosses the path of the signals coming to the mobile device on a similar bandwidth and corrupt it. An example is a microwave oven, as they operate on the 2.4GHz spectrum. This spectrum is also used by most mobile devices such as smartphones and Raspberry Pi controlled Thymio Robot.

The wireless signal is always stronger and more reliable near to a wireless access point. As shown in Figure 6.2, the mobile devices (S1 and N1) closer to the access point (N1) have a good signal level. As the mobile devices move away from the access point, the signal level degrades due to interference. The rate of packet loss increases with a decrease in the signal level. This causes latency and, therefore, affects the execution of the HMC application in case static code-offloading is used.

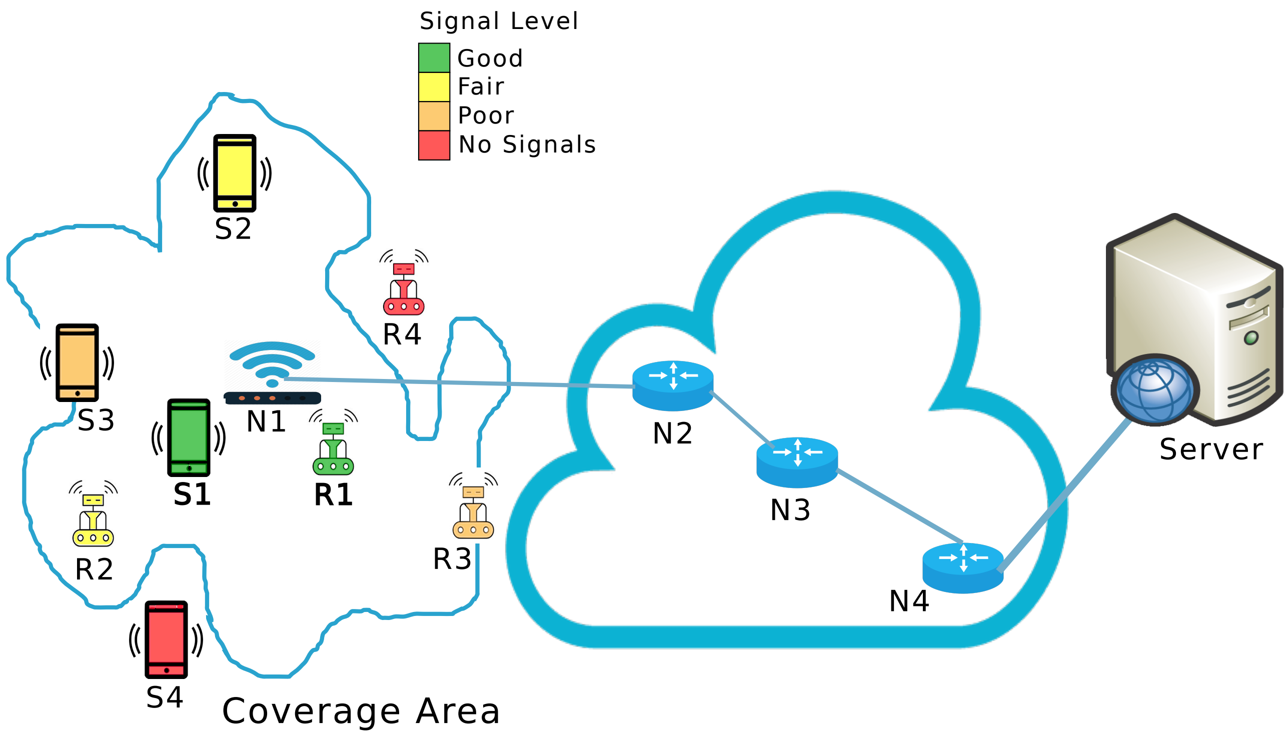


FIGURE 6.2: Mobile devices in good proximity to a wireless access point have good signals. Wireless interference degrades the signals and causes packet loss as a mobile device moves away from the access point.

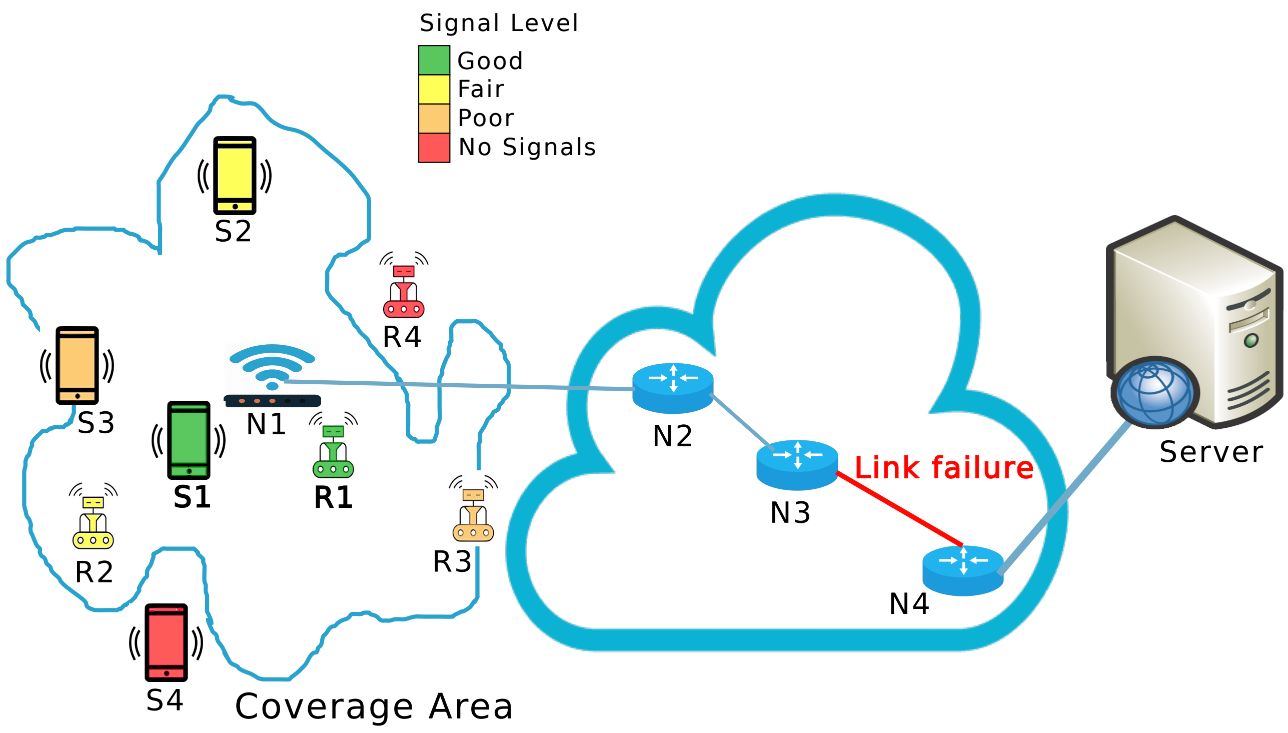


FIGURE 6.3: A link failure or network congestion can cause packet loss and latency when code-offloading is used.

Network Congestion and Network Failure

In a wireless network, a link failure can occur due to different reasons such as accidental cable cuts, a human or software error. A link failure can result in runtime errors, i.e., “the host is unreachable" exception of Java Socket library. This causes packet loss and results in latency. Other factors that can cause latency include network congestion. The congestion in a network is due to overloaded network broadcast domain where too many hosts are making requests at the same time and using the available bandwidth over its limits. Due to which packet loss happens and results in latency.

In the context of HMC applications, latency due to the network congestion or a link failure can deter the performance of the application. Mobile devices in good proximity to the access point can also suffer from network congestion or link failure. As shown in Figure 6.3, a link failure between *N3* and *N4* will affect all devices connected to *N1* that are using static code-offloading.

6.3 Case Studies

In the case studies, we will evaluate the HMC applications (ImageEffects, Mather and the Foraging task) using online profiling. We will be particularly looking into how the HMC applications perform when there are packet losses causing latency in the network, in terms of the execution time and power consumption. The applications will be executed using the following four different modes. Using no offloading, using static offloading, using self-adaptivity and using self-awareness.

1. Executing an HMC application using no code-offloading. This is to execute with all-zero configuration.
2. Executing an HMC application using static code-offloading. This is to execute with any configuration that uses one or more modules to offload to the Cloud.
3. Executing an HMC application using self-adaptive decision mechanism of the framework. The execution will switch between modes one and two during execution.
4. Executing an HMC application using self-aware decision mechanism of the framework. The execution will switch between modes one and two during execution.

6.3.1 Robotic Foraging Task

As discussed in Section 6.2.1, we created a lab-based controlled environment using which we can execute the foraging task and perform online profiling. As we can see in Figure 6.1, the robot performing the task can be accessed from the PC. An automation script that runs on the PC accessing the Raspberry Pi via SSH to perform the task and record its power consumption, network usage and its overall execution time. We consider the signal level variation, caused by factors such as wireless interference and network congestion (discussed in Section 6.2.2), as the changing environment while performing the foraging task. We operationalise the change in signal level in a Python script that executes on the Raspberry Pi.

In the script, we used a list of numerical values that represent signal levels (from good to low in dBm). The script broadcast one value each second, which represent the current signal level of the wireless network. Doing so we were able to simulate the movement of the robot in the coverage. One cycle of the signal level variation takes 60 seconds to complete. The cycle starts from a good signal level of $-32dBm$ (upper level) and decreases until a poor signal level of $-90dBm$ (lower level) is reached. It then starts increasing back to the upper level and then repeats. The data between the robot and the server is passed through the PC as shown in Figure 6.1. The PC (acting as an intermediate node) controls the flow of the packets. A script running on the PC receives the current signal level, which is broadcast from the script on the Raspberry Pi. As the signal level degrades, it causes packet loss. We imposed the packet loss at the Linux kernel-level on the PC, which is aligned with the change in the signal level. For this to achieve, we used Linux Traffic Control "*tc*" and Network Emulator "*Netem*" [9] command-line tools to randomly drop a proportion of packets.

We modelled the packet loss with respect to the signal level as: 1) from $-32dBm$ to $-70dBm$ the packet loss is 0%, 2) from $-71dBm$ to $-80dBm$ the packet loss is 20%, 3) from $-81dBm$ to $-85dBm$ the packet loss is 50% and lastly 4) from $-86dBm$ to $-90dBm$ the packet loss is 80%. We estimate these measurements by using *Wireshark* to observe the packet loss for the signal level while moving the robot away from the access point in the network coverage area.

Offline profiling at different signal levels

The offline profiling (discussed in Section 4.2) for the foraging task was carried out while keeping the robot under the footprint of good signals (averaging around $-32dBm$) from the access point. As the robot moves around the network coverage area, the signal level changes and in some cases might degrade enough to cause a large amount of packet loss.

For the self-adaptive decision mechanism to switch between no offloading and static offloading modes during the runtime, we need to find the signal thresholds for switching. We performed experiments using offline profiling at different signal levels. For each of the two Pareto efficient configurations that use code-offloading (1011 : 001000000001 and 00100000000000), we executed the foraging task for 30 independent runs each time at 7 different signal levels (mean of the signal levels) ranging from $-53dBm$ to $-90dBm$. For the all-zero configuration 00000000000000, as there is no network usage, there wont be any changes of power consumption at different signal levels. Therefore, we did not do the profiling and kept it the same as before. The average battery power consumption, network usage and execution time at these different signal levels are measured and shown in Figures 6.4a, 6.4b and 6.4c respectively.

Determining the threshold for self-adaptive switching

We can see in Figure 6.4a, the battery power consumption in case of configurations $1011 : 001000000001$ and 00100000000000 changes concerning a change in the signal level. The signal degrades from a better ($-32dBm$) to a poor ($-80dBm$) level, during which $1011 : 001000000001$ performed better than 00100000000000 and 00000000000000. Below the signal level of $-80dBm$ is an unstable network zone. In this zone, the power consumption can either increase very high (due to TCP retransmission) or decrease very low (due to latency). As shown in Figure 6.4b, with the configuration 00100000000000 the execution suffered from a long socket-wait (latency) at signal level $-87dBm$ (unstable zone). While the mobile was suffering from packet loss at the receiving end, the cloud was continuously retransmitting packets and using more network, as shown in Figure 6.4c. Similarly, execution with $1011 : 001000000001$ suffered from TCP retransmission at the signal level of $-83dBm$ in the unstable zone. This caused high delay and high network usage as shown in Figures 6.4b and 6.4c respectively.

Based on the above analysis, we choose the threshold for the self-adaptive switching at a signal level of $-80dBm$. As shown in Figure 6.4a, the static offloading configuration "$1011 : 001000000001$" performed better above $-80dBm$ and the all-zero configuration 00000000000000 is better below $-80dBm$ in terms of battery power consumption. Therefore, the self-adaptive decision mechanism of the framework while executing the foraging task will use this static offloading configuration on and above $-80dBm$ and will switch to all-zero, when the signal level is below $-80dBm$.

Online profiling: Foraging Task

As discussed in Section 6.2.2, the rate of packet loss increases due to factors like wireless interference, link failure and network congestion. The packet loss causes latency in the network and, therefore, can affect the execution of the foraging task if the mode of static code-offloading is being used. As a result, at the mobile endpoint, long socket-wait time

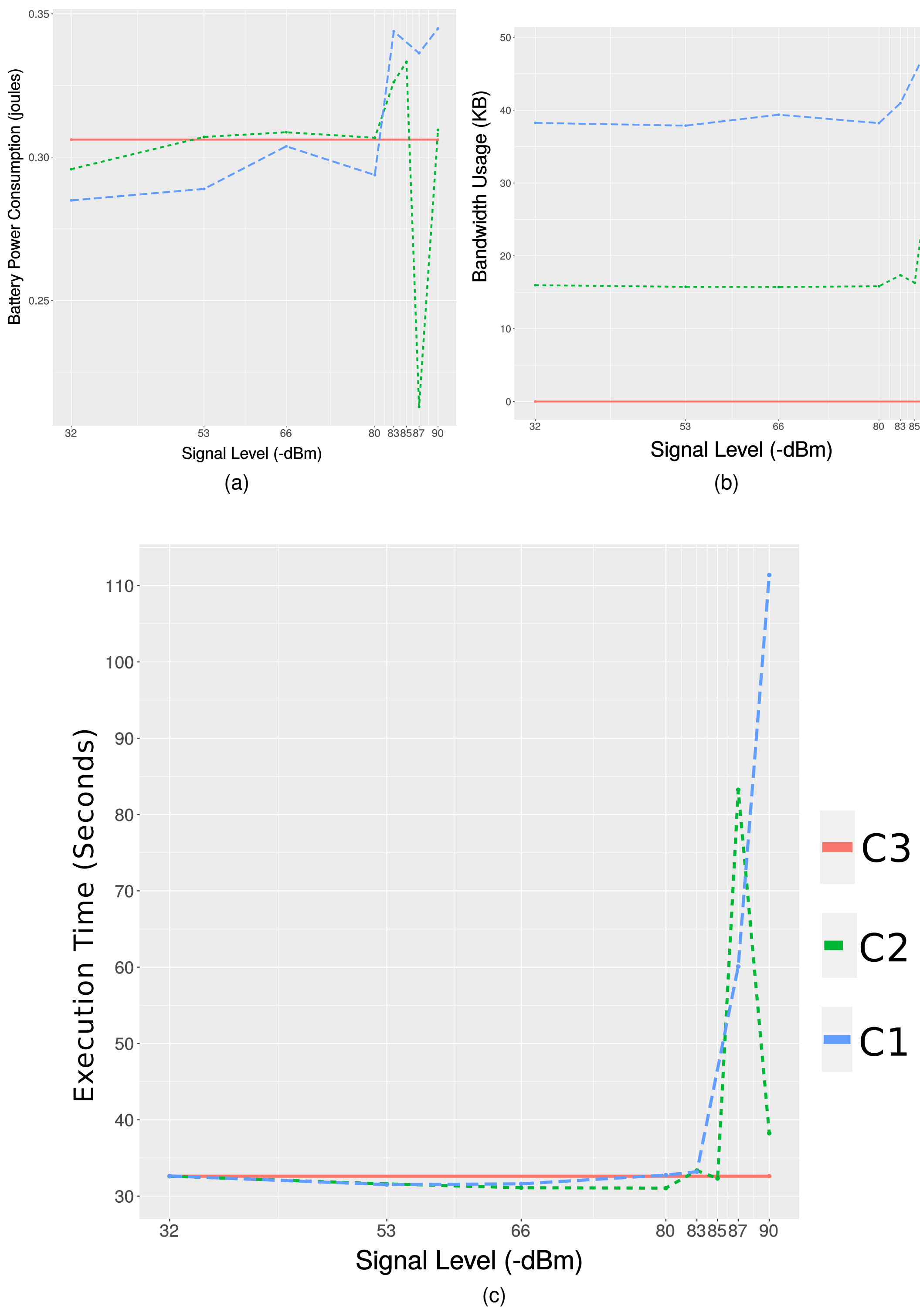


FIGURE 6.4: Plots showing the results of profiling the Pareto efficient configurations for the foraging task at different signal levels. (a) Mean power consumption of 30 runs at different signal levels. (b) Mean network usage of 30 runs at different signal level. (c) Mean execution time of 30 runs at different signal levels.

will cause more delay to the completion of the task. At the cloud, long socket-wait will cause the mobile to retransmit the TCP packets with the cost of using more battery power and network usage. We consider the following two scenarios, which are intended to capture two contrasting types of environment that might be encountered by a robot.

1. In the first scenario, we will consider zero congestion in the network. As shown in Figure 6.5, the congestion is kept at zero during the task execution.
2. In the second scenario, we introduce a high degree of network congestion (100%) at certain times, when the packet loss is 100%, as shown in Figure 6.7. During the congestion, the self-adaptive approach will not switch to all-zero configuration, as the signal level would still be good (from $-50dBm$ to $-70dBm$).

Results and Analysis

To evaluate the performance of the runtime decision mechanism of the framework using self-adaptivity and self-awareness, we executed the foraging task for an increased time duration compared to the offline profiling. The task was executed on the robot and during the runtime the proposed self-aware and self-adaptive approaches were used by the task. The corresponding battery consumption cost has been included in the results. For each of the two scenarios discussed previously, we obtained results for the following four modes of execution.

1. Using no code offloading, which is executing the task with all-zero configuration "$C3$".
2. Using static code offloading, which is executing the task with the configuration that uses code-offloading "$C1$".
3. Using self-adaptive approach of the framework.
4. Using self-aware approach of the framework.

The measurements for the two online scenarios using the four execution modes are listed in Table 6.1. The execution of the task with $C3$, which does not use code-offloading,

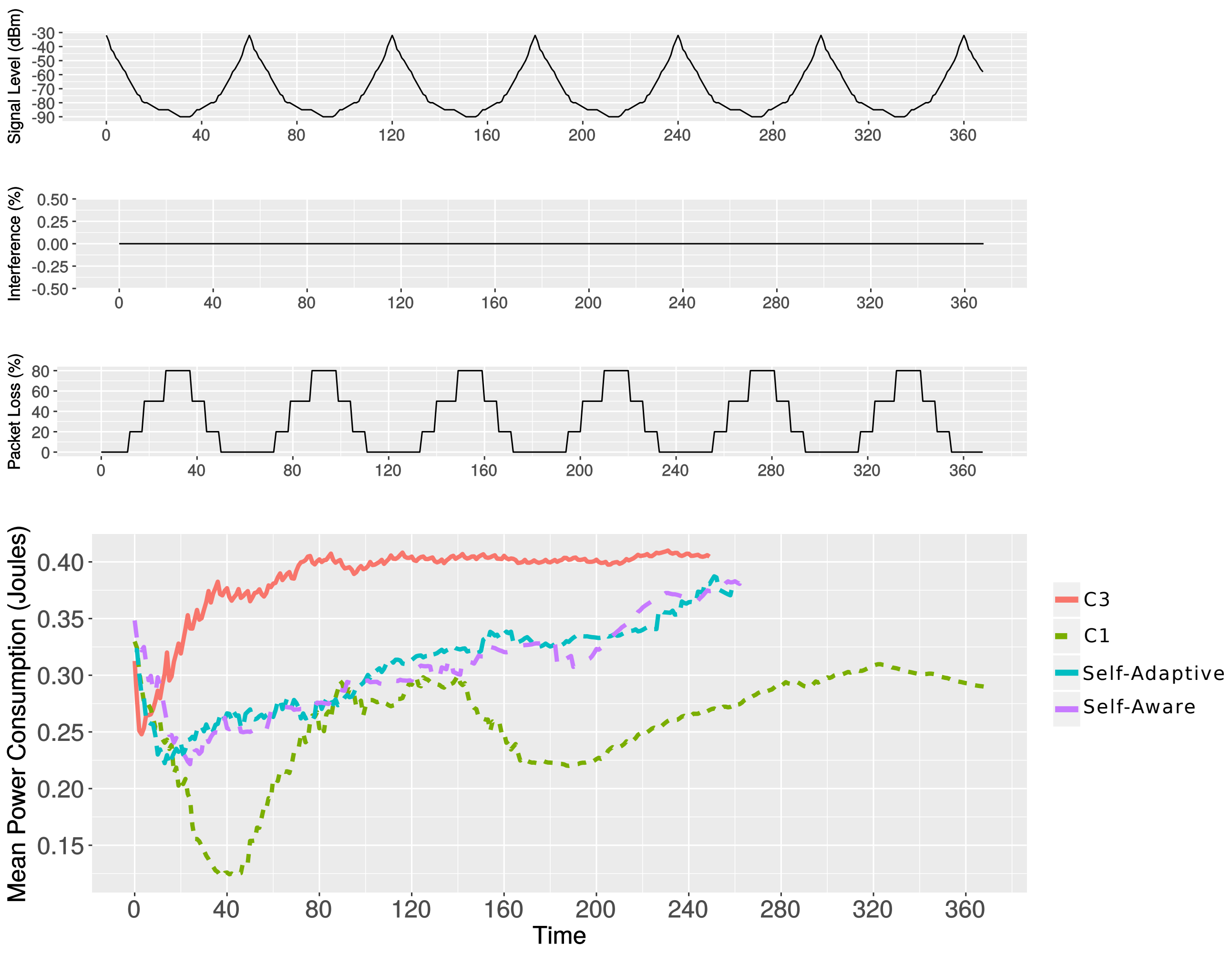


FIGURE 6.5: Plots showing results of the first scenario in which the robot moves around the coverage area of the wireless network and perform the foraging task. While roaming, the signal level changes because of wireless interference and causing packet loss and resulting latency. In this scenario, we are ignoring packet loss due to other factors like network congestion or link failure. The cumulative mean of battery power consumption during the execution is plotted over time. The lines end show the completion of the task. The modes of execution using the self-adaptive and self-aware decisions achieve better optimisation in a changing environment (signal level variation). The mode of execution using static code-offloading $C1$ is affected by network latency. The mode of execution using no code-offloading $C3$ used more power to complete the task.

is used as the same in both scenarios. We can see that the network usage was zero in this execution. The executions of the task with $C1$, which uses code-offloading, suffered from high latency in both scenarios. The network usage was high, which shows that extra data was used due to latency or congestion that was dropping the packets. In the two online profiling scenarios, the execution of the task was limited to only one. Therefore, as we can see in Table 6.1, the mean values of power and network usages are from only one

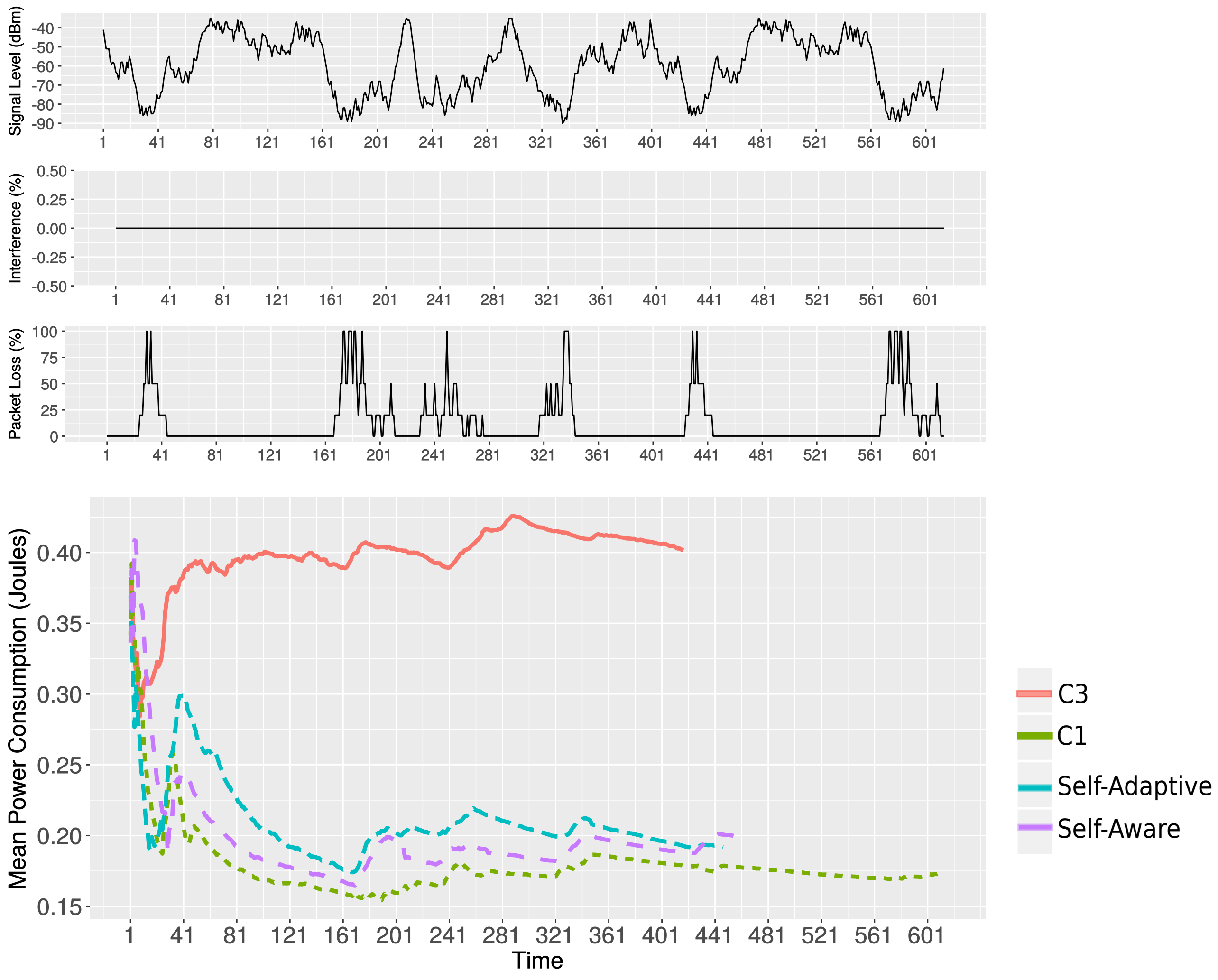


FIGURE 6.6: Plots showing results of the first scenario in which the robot moves around the coverage area of the wireless network and perform the foraging task. While roaming, the signal level changes in a random pattern because of wireless interference and causing packet loss and resulting latency. In this scenario, we are ignoring packet loss due to other factors like network congestion or link failure. The cumulative mean of battery power consumption during the execution is plotted over time. The lines end show the completion of the task. The modes of execution using the self-adaptive and self-aware decisions achieve better optimisation in a changing environment (signal level variation). The mode of execution using static code-offloading $C1$ is affected by network latency. The mode of execution using no code-offloading $C3$ used more power to complete the task.

sample of their measurements. Also, we can see that the standard deviations values are zero.

Furthermore, we have used two different lab-based setups to execute the foraging task. In the first setup, the signal level degrades and improve in a continues order. This

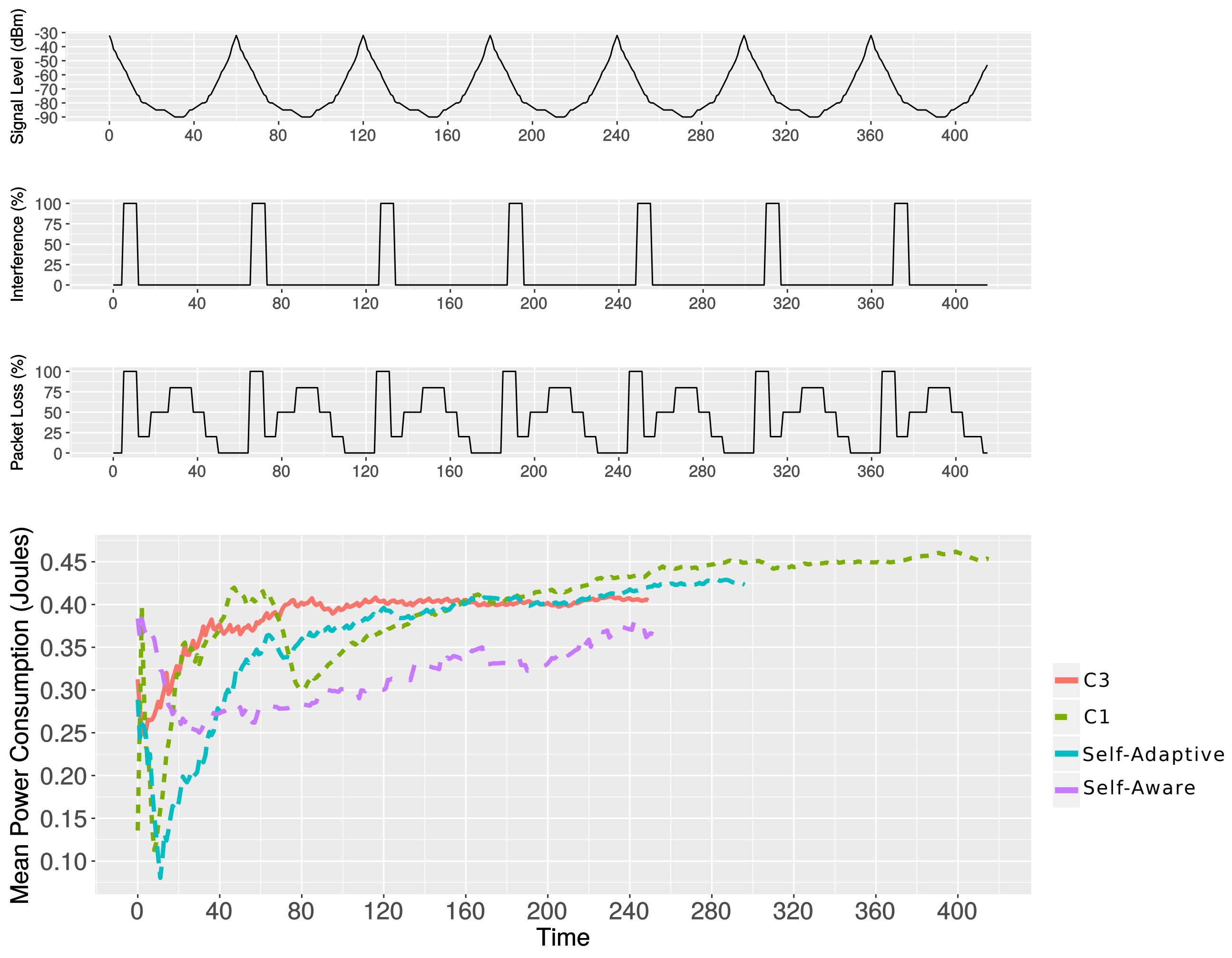


FIGURE 6.7: Plots showing results of the second scenario in which the robot moves around the coverage area of the wireless network and perform the foraging task. While roaming, the signal level changes because of wireless interference and causing packet loss. Also, we introduce packet loss of 100% that is caused by factors such as congestion in the network or a link failure. The cumulative mean of battery power consumption during the execution is plotted over time. The lines end show the completion of the task. The mode of execution using the self-aware decision mechanism out-performed the self-adaptive, $C1$ and $C3$ modes of executions in terms of using less power and complete in less time. It is because the robot monitored the packet loss by observing its runtime impact i.e., avoiding latency.

represents that the mobile device is moving away and then coming towards the WIFI access point, while there are no other obstacles in the middle to cause interference. In the second setup, the signal level drops and improve randomly on the receiving mobile device. This represents that there are obstacles in the middle causing interference. Moreover, the simulation time was further increased in the case of random signals because the application also takes time to switch between configurations. Since the signals were

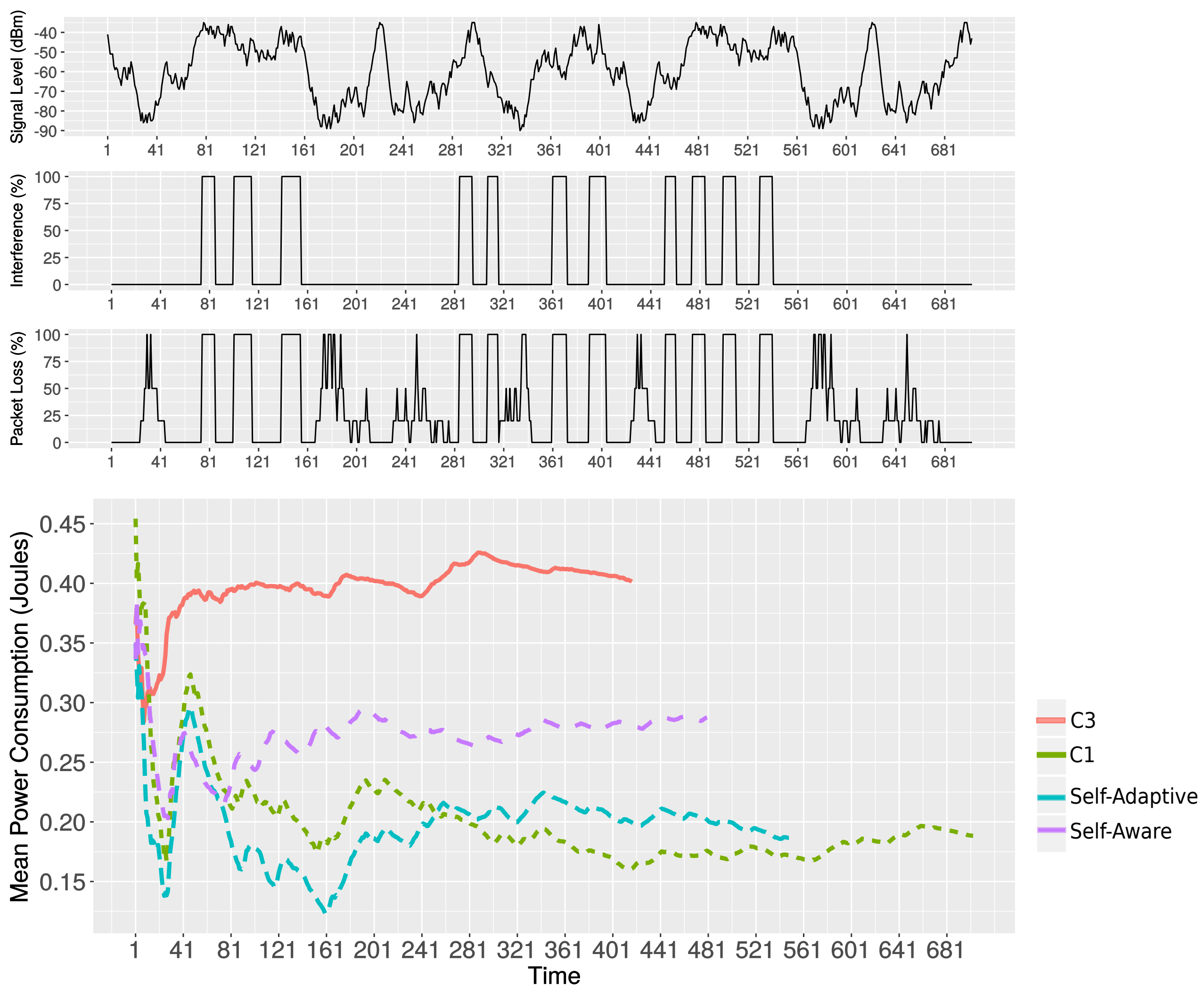


FIGURE 6.8: Plots showing results of the second scenario in which the robot moves around the coverage area of the wireless network and perform the foraging task. While roaming, the signal level changes randomly because of wireless interference and causing packet loss. Also, we introduce packet loss of 100% that is caused by factors such as congestion in the network or a link failure. The cumulative mean of battery power consumption during the execution is plotted over time. The lines end show the completion of the task. The mode of execution using the self-aware decision mechanism out-performed the self-adaptive, $C1$ and $C3$ modes of executions in terms of using less power and complete in less time. It is because the robot monitored the packet loss by observing its runtime impact i.e., avoiding latency.

dropping/improving randomly, the number of switching were high in this case.

In scenario one, the mode of execution using the self-adaptive or self-aware decision mechanisms resulted with consuming less battery power (with the cost of using the network) compare to the mode in which no code offloading is used $C3$. This is shown in

TABLE 6.1: Offline and online profiling of a mobile-cloud hybrid foraging task performed by a Rasberry Pi controlled Thymio robot. The number of executions per Pareto efficeint configurations and the mean and standard deviation of battery power consumption, network usage and execution time are stated.

Profiling	**Configuration**	**Granularity Level**	**Executions**	**Execution Time (secs)**	**Battery Power Consumption (Joules)**		**Network Usage (kB's)**	
				Mean	Mean	Standard Deviation	Mean	Standard Deviation
Offline	C1 = 1011:001000000001	Hybrid	30	32.6	0.2849	0.0113	38.2503	0.0695
	C2 = 00100000000000	Method	30	32.6	0.2958	0.018	15.9743	0.0971
	C3 = 00000000000000	Method	30	32.6	0.3061	0.0136	0	0
Online (Scenario One)	C3	Method	1	251	0.4046	0	0	0
	C1	Hybrid	1	368	0.29	0	486.539	0
	Self-adaptive (C1, C3)	Hybrid, Method	1	259	0.3793	0	192.117	0
	Self-aware (C1, C3)	Hybrid, Method	1	264	0.3885	0	169.51	0
Online (Scenario Two)	C3	Method	1	251	0.4046	0	0	0
	C1	Hybrid	1	415	0.4529	0	502.065	0
	Self-adaptive (C1, C3)	Hybrid, Method	1	298	0.4232	0	312.917	0
	Self-aware (C1, C3)	Hybrid, Method	1	255	0.3733	0	163.314	0

Figures 6.5 6.6. Using self-adaptive and self-aware decisions during runtime the executions did not suffer from latency as happened in the case of $C1$, which is executing the task with static code-offloading. This is because the framework would switch to all-zero configuration ($C3$) when the signal level was low and causing packet loss. Therefore, execution with static code offloading using configuration $C3$ took more time to complete.

In scenario two, the execution of the task with the self-adaptivity or self-awareness again performed better than $C1$ by taking less time to complete. However, with the self-adaptivity, the execution suffered from latency due to packet loss caused by the network congestion as shown in Figures 6.7 and 6.8. The mode of execution with the self-awareness performed better by avoiding the packet loss caused by both network congestion and low signal level due to interference. The self-aware decision mechanism consumed less battery power than $C3$ and less network usage than $C1$ and the self-adaptive, and also finished in good time.

It should be noted that in the lab setup that uses random signal drop/improve (Figures 6.6 and 6.8) the power consumption of executions using self-adaptive and self-aware approaches is very less than the lab setup where the signals drop/improve were continues (Figures 6.5 and 6.7). This is because the executions were mostly using code offloading as there were very occasional time slots where the packet loss was more than 50%.

6.3.2 Android Application: ImageEffects

We implemented the self-adaptive and self-aware decision mechanisms in our Android-based MC hybrid framework. The aim was to evaluate how they can perform on Android-based mobile devices in terms of avoiding network latency and battery power consumption. We performed online profiling for the ImageEffects (the Android-based MC hybrid application). As shown in Figure 6.1, a mobile device, i.e. an Android-based smartphone connected to a PC via USB cable, can be accessed from the PC to automate the execution of the ImageEffects installed on the mobile device. A Python-based script, running on the PC, automate the execution of ImageEffects and controls the packets flow from the smartphone to the server passing through the PC (an intermediate node). We used the Motorola G4 Android smartphone for the experimental study. As discussed in Section 4.1.1, the Monitor application was employed to measure battery power consumption, network usage and execution time of one complete run of the ImageEffects.

To simulate the change in the WiFi signal level as the mobile device moves in the coverage area of a wireless network, we created an application that runs on the mobile device. The application has a cycle of signal levels starting from a good level and going down to a weak signal level. This application broadcast the signal levels per second. The Monitor and ImageEffects applications can receive the broadcasts using the Android's built-in Broadcast Receiver. Doing so we were able to profile the network latency. In the experimental study, we evaluated the runtime behaviour of ImageEffects corresponding to network latency caused by wireless interference and congestion.

Determining the threshold for self-adaptive switching

In Chapter 4, we discussed the offline profiling of ImageEffects, which was done under the footprint of good signals (averaging around $-30dBm$) from the access point. However, as the signal level decreases, the network becomes unstable which can affect the execution of MC hybrid applications when static offloading is used. Therefore, the self-adaptive decision mechanism required the signal threshold. Below the threshold, the execution will switch to all-zero configuration and will not be using the code-offloading. To determine the threshold of signal level for the adaptive switching, we carried out offline profiling at three different signal levels ranging from good to poor.

TABLE 6.2: Offline and online profiling of the mobile-cloud hybrid Android-based application: ImageEffects. The number of executions per configurations and the mean and standard deviation of battery power consumption, network usage and execution time are stated.

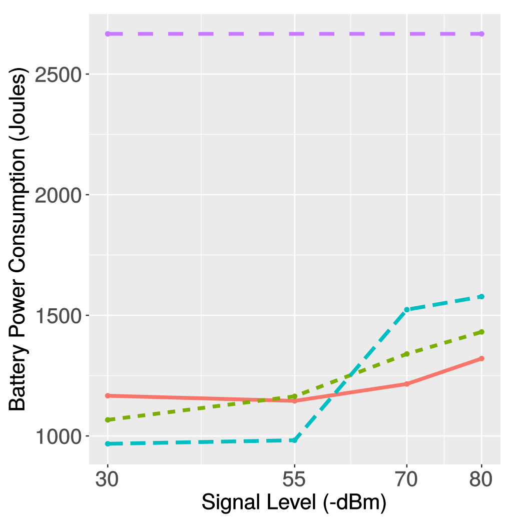
Profiling	**Configuration**	**Granularity Level**	**Executions**	**Execution Time (secs)**	**Battery Power Consumption (Joules)**		**Network Usage (kB's)**	
				Mean	Mean	Standard Deviation	Mean	Standard Deviation
Offline	zero = 1010:00000000	Hybrid	30	10.03	2666.68	587.5	133.0	0.8
	one = 1000:010000	Hybrid	30	8.13	1166.69	267.0	669.7	1.3
	two = 1000100000	Method	30	8.09	967.57	279.3	1201.3	1.5
	three = 1011:110000100	Hybrid	30	8.02	1066.7	308.9	1065.7	1.4
Online (Scenario One)	zero	Hybrid	1	514	245.78	0	14529.95	0
	one	Hybrid	1	491	257.74	0	75608.73	0
	two	Method	1	542	267.1	0	82570.3	0
	three	Hybrid	1	667	361.19	0	119269.87	0
	Self-adaptive (zero, one, three)	Hybrid	1	446	239.37	0	67171.84	0
	Self-aware (zero, one, three)	Hybrid	1	458	235.51	0	70142.41	0
Online (Scenario Two)	zero	Hybrid	1	514	245.78	0	14529.95	0
	one	Hybrid	1	605	290.96	0	78170.29	0
	two	Method	1	704	304.47	0	93699.91	0
	three	Hybrid	1	598	279.18	0	73114.5	0
	Self-adaptive (zero, one, three)	Hybrid	1	565	226.35	0	74042.41	0
	Self-aware (zero, one, three)	Hybrid	1	436	216.53	0	60669.72	0

As shown in Figure 6.9, we have plotted the battery power consumption, network usage and the execution time of the offline profiling at different signal levels. The four Pareto efficient configurations (listed in Table 6.2) of ImageEffects were used. The measurements from all-zero configuration (zero) were kept the same as before. We can see in Figure 6.9a that the zero configuration has the highest power consumption at all signal levels but has less network usage. Also, the execution with zero configuration will not be affected by the network latency as shown in Figure 6.9c. On the other hand, as the signals become poorer the execution with configuration "two" become more expensive in terms of using the battery power. In the unstable network zone (below $-80dBm$), the execution time with configuration "two" was more than the configurations using "one" and "three" modules to offload to the cloud. Based on this analysis, we have chosen the following switching thresholds for the self-adaptive decision mechanism for the Android-based MC framework.

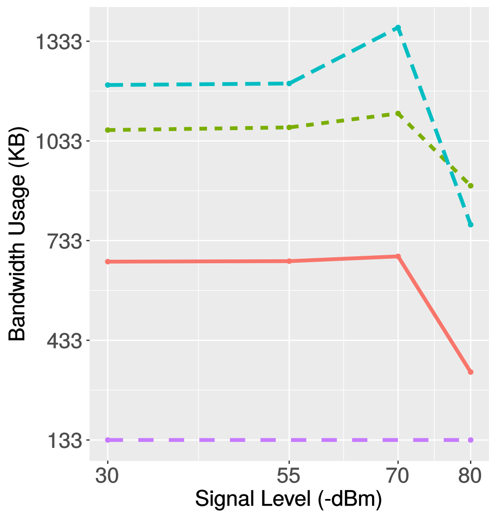
1. In the coverage area when the signals are robust, above $-60dBm$, the framework will use the static code-offloading execution mode by employing configuration "three".

2. The framework will use configuration "one" at and below $-60dBm$.

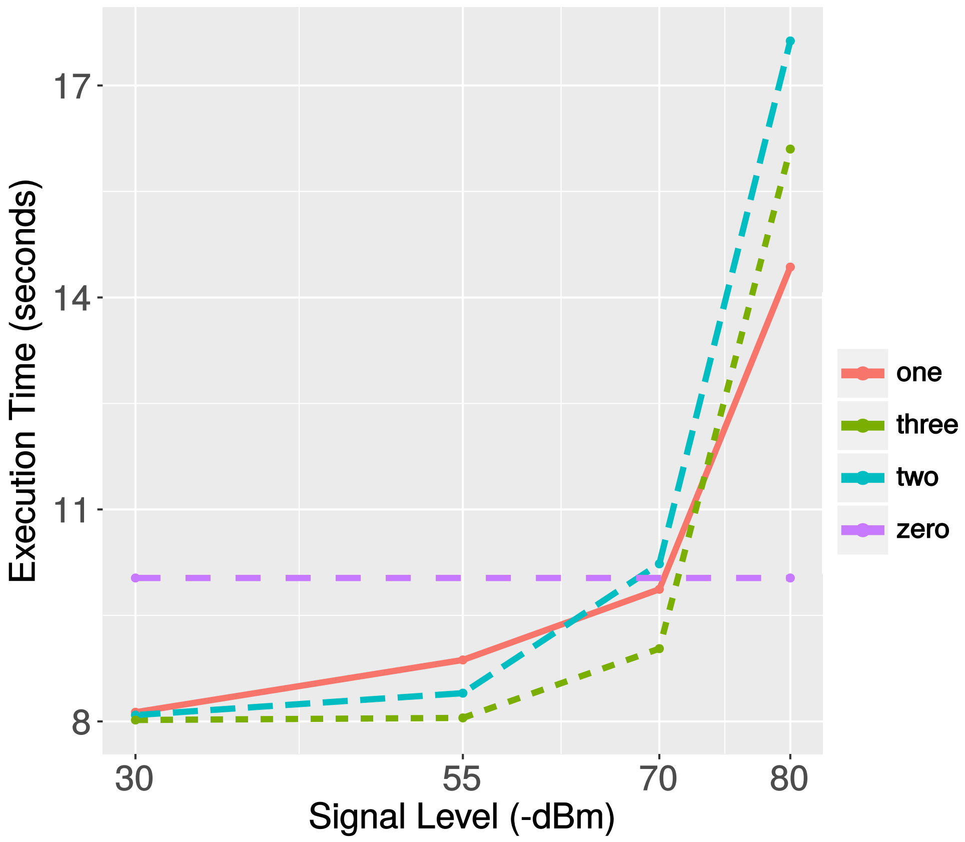
3. In the unstable network zone, below $-80dBm$, the framework will use configuration "zero" to execute the tasks of ImageEffects.



(a)



(b)



(c)

FIGURE 6.9: Plots showing the results of profiling the Pareto efficient configurations at different signal levels. (a) Cumulative power consumption of 30 runs. (b) Cumulative network usage of 30 runs. (c) Cumulative execution time of 30 runs.

Online profiling: ImageEffects

We discussed network latency caused by factors such as wireless interference and network congestion in Section 6.2.2. To evaluate the runtime behaviour of ImageEffects in the presence of network latency, we will use the lab-based controlled environment for the online profile. We increased the execution time of ImageEffects completing the tasks. We consider the following two scenarios, which are intended to capture two contrasting types of environment that might be encountered by smartphone.

1. In the first scenario, we will consider zero per cent congestion in the network. As shown in Figure 6.10, the congestion is kept zero during the application execution.
2. In the second scenario, we introduce a high degree of network congestion (100%), where the packet loss is 100%, as shown in Figure 6.11. During the congestion, we assume that the self-adaptive approach will likely be unable to switch to all-zero configuration as the signal level would still be good (from $-60dBm$ to $-65dBm$).

Results and Analysis

In the online profiling, the battery power consumption, network usage and execution time for both scenarios were measured and listed in Table 6.2. The ImageEffects was executed on the mobile device and during the runtime the proposed self-aware and self-adaptive approaches were used. The corresponding battery consumption cost has been included in the results. The execution with configuration “zero", which does not use code-offloading, is used as the same in both scenarios. In both scenarios, we obtained results for the following four modes.

1. Using no code offloading, which is executing the task with the “zero" configuration.
2. Using static code offloading, which is executing the task with the configurations that offload “one", “two" and “three" modules.
3. Using self-adaptive approach of the Android-based framework.
4. Using self-aware approach of the Android-based framework.

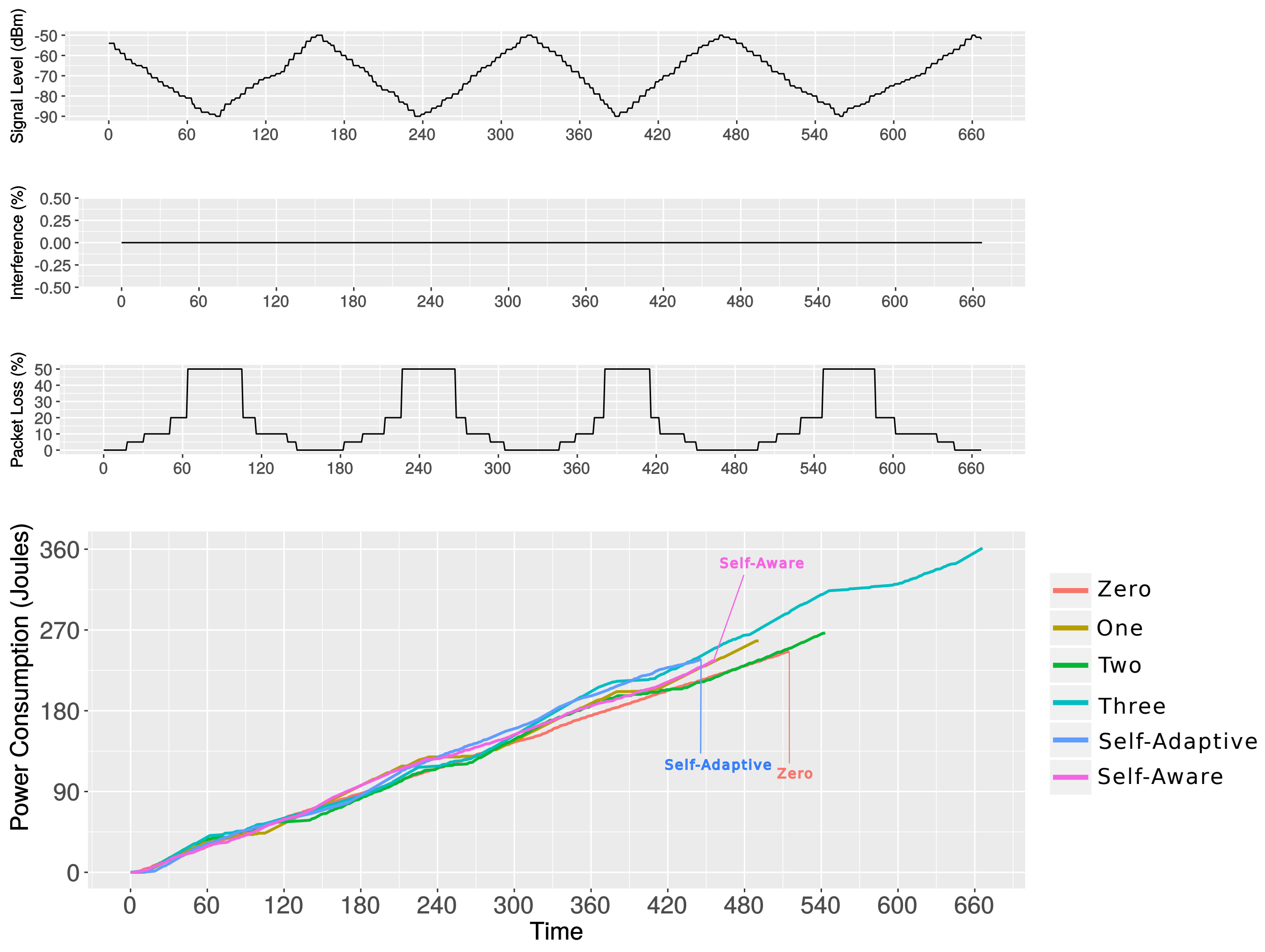


FIGURE 6.10: Plots showing results of the first scenario in which Image-Effects, an Android-based MC hybrid application, is used while moving around the coverage area of a wireless network. During runtime, the signal level changes due to wireless interference and, therefore, causing packet loss that results in latency. In this scenario, we are ignoring packet loss due to other factors such as network congestion or link failure. Cumulative battery power consumption during the execution is plotted over time. The lines end show the completion of the task. The modes of execution using the self-adaptive and self-aware decisions achieve better optimisation in a changing environment (signal level variation) in terms of completion time. The mode of execution using static code-offloading “two" and “three" were greatly affected by network latency. The mode of execution using no code-offloading “zero" used nearly the same power as self-adaptive and self-aware, but take more time to complete.

We can see in Figure 6.10 that in the case of scenario one, the self-adaptive and self-aware both were completed in less time than execution with “zero" configuration. These executions consumed nearly similar battery power. The mode of executions using static code offloading was affected by wireless interference.

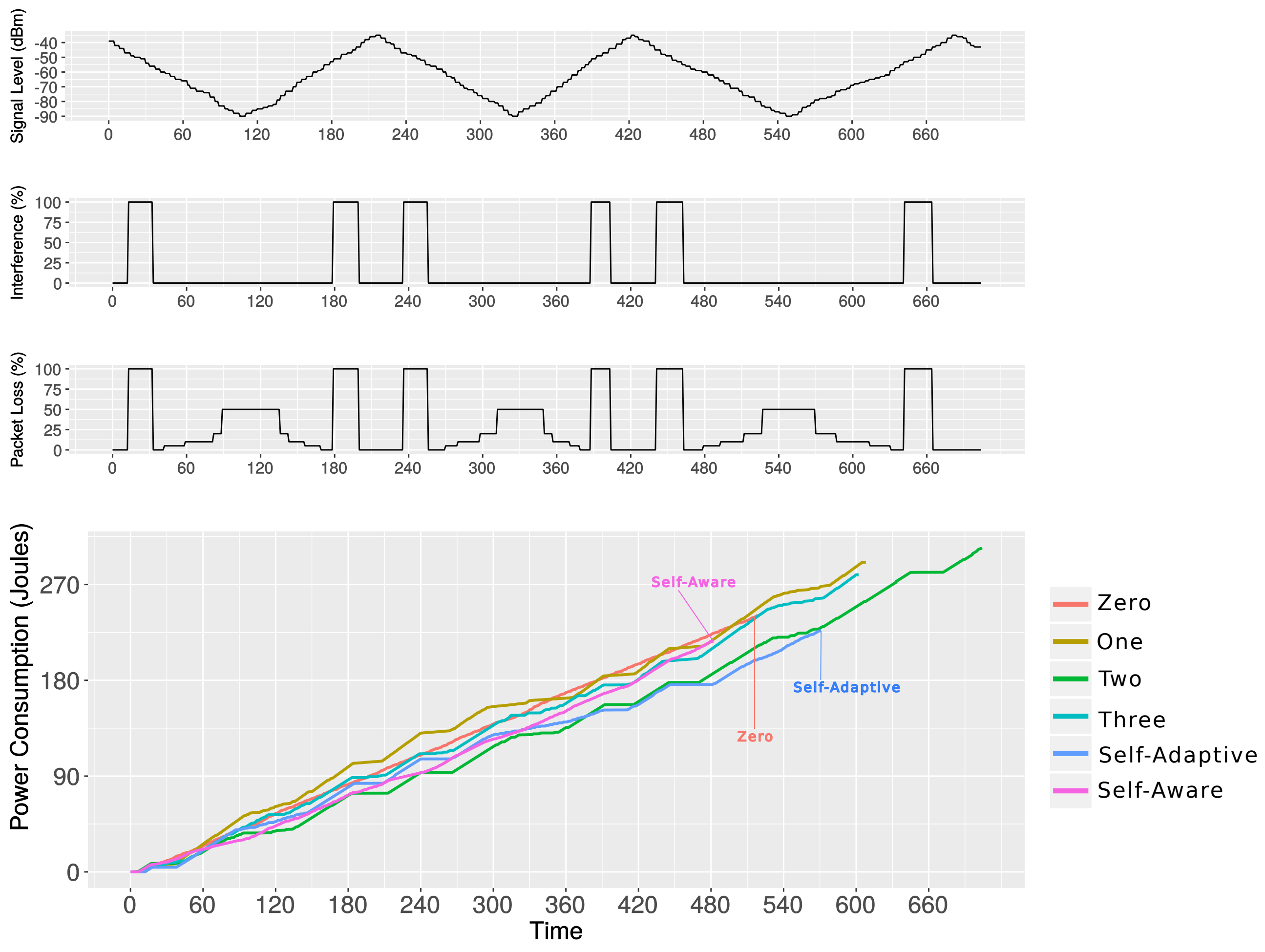


FIGURE 6.11: Plots showing results of the second scenario in which Image-Effects, an Android-based MC hybrid application, is used while moving around the coverage area of a wireless network. While roaming, the signal level changes because of wireless interference and causing packet loss. Also, we introduce packet loss of 100% that is caused by factors such as congestion in the network or a link failure. Cumulative battery power consumption during the execution is plotted over time. The lines end show the completion of the task. The mode of execution using the self-aware decision mechanism out-performed the other modes of executions in terms of avoiding latency and completed in less time, as it monitored the packet loss by observing its runtime impact.

In scenario two, the execution mode using the self-aware decision mechanism performed better than others in terms of avoiding the network latency caused by interference. The execution mode using self-adaptive decision mechanism suffered from latency due to packet loss caused by the network congestion as shown in Figure 6.11.

TABLE 6.3: Offline and online profiling of the mobile-cloud hybrid Android-based application: Mather. The number of executions per configurations and the mean and standard deviation of battery power consumption, network usage and execution time are stated.

Profiling	**Configuration**	**Granularity Level**	**Executions**	**Execution Time (secs)**	**Battery Power Consumption (Joules)**		**Network Usage (kB's)**	
				Mean	Mean	Standard Deviation	Mean	Standard Deviation
Offline	zero = 01:0000	Hybrid	100	20.06	16.75	3.1	0	0
	one = 001000	Method	100	20.12	14.74	2.97	3.07	0.03
	two = 10:0110	Hybrid	100	20.07	10.89	2.75	3.73	0.05
Online (Scenario One)	zero	Hybrid	1	492	37.53	0	0	0
	one	Method	1	520	35.62	0	58.94	0
	two	Hybrid	1	515	44.09	0	129.16	0
	Self-adaptive (zero, one, two)	Hybrid and Method	1	490	34.74	0	92.19	0
	Self-aware (zero, one, two)	Hybrid and Method	1	496	33.64	0	108.07	0
Online (Scenario Two)	zero	Hybrid	1	808	60.2	0	0	0
	one	Method	1	1046	77.73	0	117.99	0
	two	Method	1	1053	78.12	0	249.13	0
	Self-adaptive (zero, one, two)	Hybrid and Method	1	963	79.79	0	198.42	0
	Self-aware (zero, one, two)	Hybrid and Method	1	797	57.51	0	197.86	0

6.3.3 Android Application: Mather

As discussed in the previous case study (ImageEffects), we used the same workflow for Mather to perform online profiling. We consider the same two lab-based scenarios. In the first scenario, the wireless interference caused the number of packet loss due to latency as shown in Figure 6.12. In the second scenario, the congestion in the network along with the wireless interference caused the number of packet loss due to network latency as shown in Figure 6.13. We increased the time of execution in order to understand the effect of different network condition with time. In both scenarios, the execution of Mather was done for the four modes of the MC framework. 1) Using no code-offloading (zero). 2) Using static code-offloading (one and two). 3) Using the self-adaptive decision mechanism. 4) Using the self-aware decision mechanism. The measurements recorded for both scenarios are listed in Table 6.3.

Results and Analysis

The Mather was executed on the mobile device and during the runtime the proposed self-aware and self-adaptive approaches were used. The corresponding battery consumption cost has been included in the results.

In scenario one, the modes of executions using zero configuration, self-adaptive and self-aware completed in nearly the same time as can be seen in Figure 6.12. These executions were not affected by the latency. In scenario two, the self-aware mode of execution outperformed all others in terms of using less battery power. It also completed in less time than others, as shown in Figure 6.13.

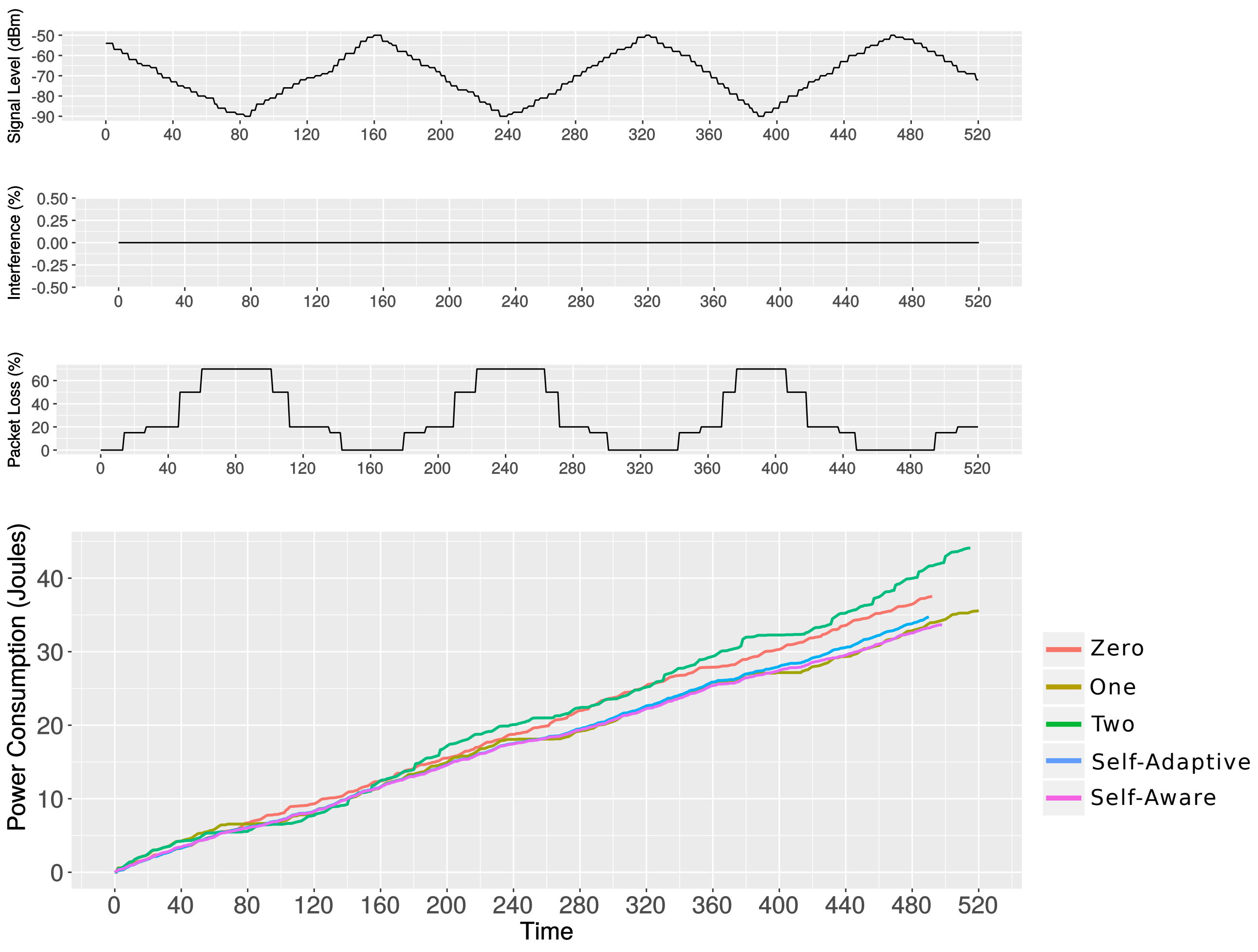


FIGURE 6.12: Plots showing results of the first scenario in which Mather, an Android-based HMC application, is used while moving around the coverage area of a wireless network. During runtime, the signal level changes due to wireless interference and, therefore, causing packet loss that results in latency. In this scenario, we are ignoring packet loss due to other factors such as network congestion or link failure. Cumulative battery power consumption during the execution is plotted over time. The lines end show the completion of the task. The modes of execution using the self-adaptive and self-aware decisions achieve better optimisation in a changing environment (signal level variation) in terms of both power consumption and completion time. The mode of execution using static code-offloading "one" and "two" were affected by network latency. The mode of execution using no code-offloading "zero" completed in nearly the same time as self-adaptive and self-aware, but it consumed more power.

6.4 Limitations of the Approach

In this chapter, we used self-adaptive and self-aware decision mechanism of the HMC application framework to make runtime decisions. In the case studies, we discussed experimental results in which we used online profiling. We discuss some of the limitations

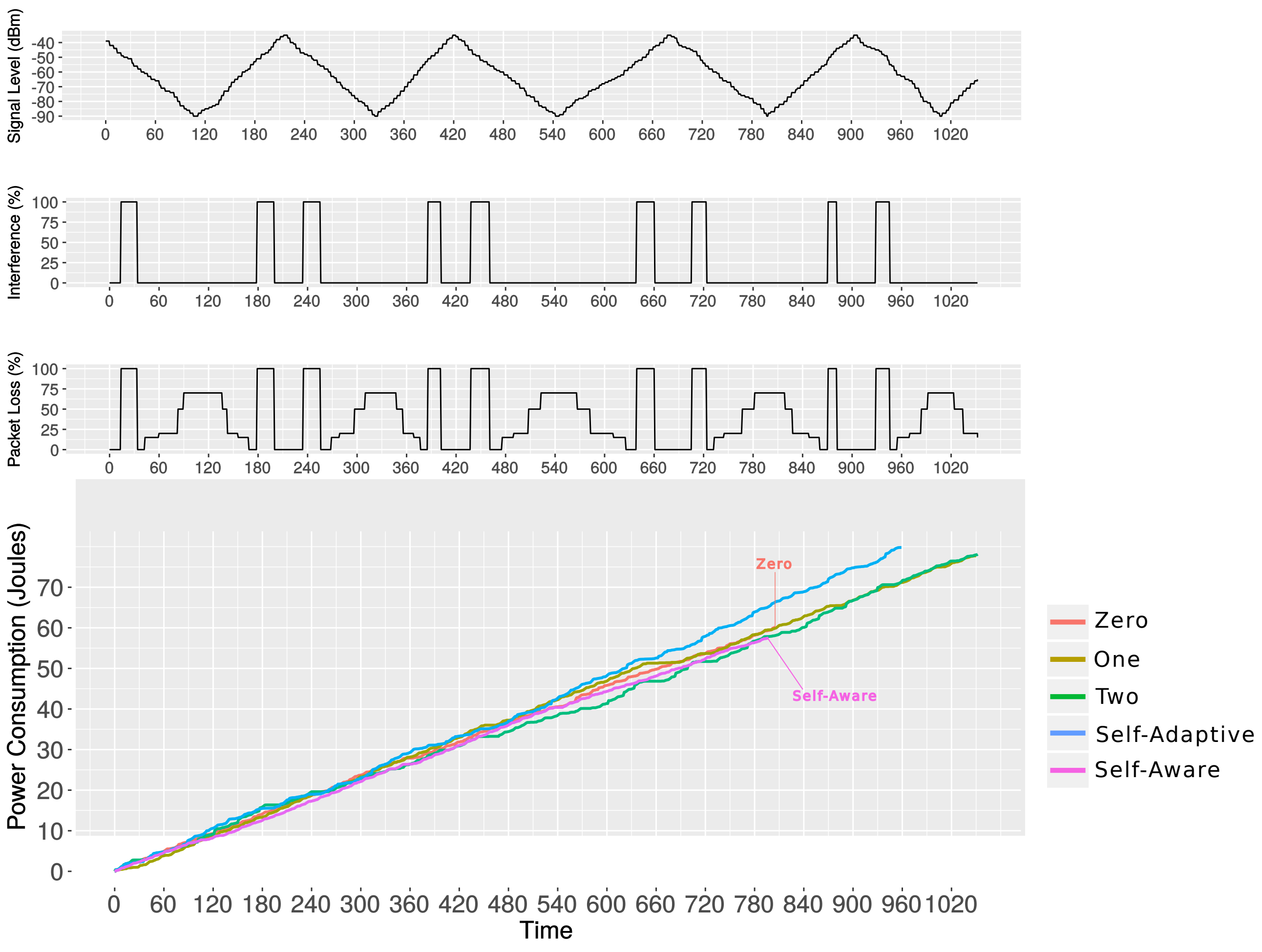


FIGURE 6.13: Plots showing results of the second scenario in which Mather, an Android-based HMC application, is used while moving around the coverage area of a wireless network. During runtime, the signal level changes because of wireless interference and causing packet loss. Also, we introduce packet loss of 100% that can be caused by factors such as congestion in the network or a link failure. Cumulative battery power consumption during the execution is plotted over time. The lines end show the completion of the task. The mode of execution using the self-aware decision mechanism out-performed all other modes of executions in terms of avoiding latency and completed in less time, as it monitored the packet loss by observing its runtime impact.

of the approach as following.

1. The self-adaptive and self-aware decision mechanisms only take into consideration the signal level and packet loss as the changing factors. Other factors such as CPU usage can added in the future work. For example, when the OS is using too much processing power for system related tasks, the decision mechanism would decide to offload the code to the cloud.

2. During execution of the HMC applications, in the lab environment,To validate the idea of energy-efficient, scalable, self-adaptive and self-aware mobile-cloud hybrid computing systems, we presented our technique and discussed the experimental results. However, this work can be extended in some areas. We discuss some future direction as follow.

   In this work, we have considered applications developed using Object Oriented Programming paradigm for mobile devices. We modularise their code based on classes and methods. We create configurations of applications which are coarse-grained (class-level) or fine-grained (method-level). However, there are other types of programming paradigms that are also used to develop application software, i.e., functional programming and procedural programming. Moreover, cross-platform mobile applications are developed using HTML, Javascript and XML etc, where the concept of OOP (i.e., classes) is not followed. This is one area that is strongly recommended to be explored in the future.

   Secondly, in this work, as we have mentioned in Chapter 3, the programmers of an HMC application annotate the code manually during the development time. This is done to identify offloadable modules in the code. A converter application is then used to analyse the source code and inject the offloading interface for the modules that were annotated. In future work, static analysis of the applications can be used to automatically find which modules are suitable for code offloading. The computationally-intensive modules can be identified by techniques such as the number of lines of code in a module, data types used in a module, the number of times a loop will execute etc.

   Thirdly, we have used multi-objective optimisation to minimise two objectives, battery power consumption and network usage, using offline profiling. The future work could extend the multi-objective optimisation to more than two objectives.

For example, the objectives of minimising delay and maximising application's performance could be added to offline profiling. Moreover, the experimental work using offline profiling can be extended to other underlying technologies, e.g., mobile-edge and mobile-fog. Currently, we have only considered mobile-cloud as the underlying technology.

Fourthly, the Two-Step search algorithm, which we proposed in this work, converges in a feasible amount of time for small to medium size applications. For large scale applications, where the offloadable class-level and method-level modules are higher than 8, it would not be practical to use the Two-Step algorithm. This is because the exhaustive search in step 1 would not complete in a feasible amount of time. Instead, a fast search algorithm such as NSGA-II can be used in future work.

Fifthly, the self-adaptive and self-aware decision mechanisms only take into consideration the signal level and packet loss as the changing factors. Other factors such as CPU usage can be added in the future work. For example, when the Operating System is using too much processing power for system-related tasks, the HMC application will switch to a configuration that offloads the code to the cloud. Furthermore, in future work, a formal method/algorithm can be implemented for the self-adaptive and self-aware approaches. Also, the framework can be evaluated with other contextual factors such as resource usage, code type, cloud-side context and using a non-WIFI network (e.g., LoRa/LoRaWAN, NB-IoT, SigFox) we have used a WIFI network, where the signal level is monitored by the HMC applications. In situations such as a mobile device is tethered to another device via a cable to use its network, the self-adaptive decision mechanism of the framework will not work.

6.5 Summary of Contributions

In this chapter, we discussed a method to optimise the efficiency trade-off and avoid network latency during runtime of HMC applications created for Android-based smartphones and robots operated by Linux-based computing systems. The runtime decisions are made using self-adaptivity and self-awareness in the HMC framework. Using a workflow for online profiling of the HMC applications, we were able to simulate the change in the operating environment in a lab-based controlled setup.

The self-adaptive switching was based on an evaluated signal level threshold. The mobile devices (Android smartphone and Thymio robot) executing the HMC applications were able to switch between the Pareto efficient configurations. When the signal level was good, the framework would switch to a configuration that executes maximum number of modules on the cloud. In case of weak signal level, the framework would switch to a configuration that executes all modules on the device.

The self-aware decision mechanism was based on monitoring the packet loss by the mobile device within itself instead based on the change of the signal level. With a high amount of packet loss, the framework would switch to the Pareto efficient configuration that would execute all modules on the device. With zero to a small amount of packet loss, the framework would switch to other configurations that use code-offloading.

The analysis of our experimental study provided some key facts regarding online optimisation. Some key findings in line with the main contributions are enumerated as follow.

1. By simulating a scenario in which a mobile device was moving into the network coverage area experiencing a change in signal level. We observed that using the self-adaptive and self-aware decisions performed better than using static offloading and no offloading, in terms of completion time and optimising the efficiency trade-off.

2. By simulating a scenario in which a mobile device was moving into the network coverage area experiencing: 1) a change in signal level, and 2) network latency caused due to congestion in the network. We observed that the self-aware decision mechanism performed better than the self-adaptive, static offloading, and no offloading - in terms of optimising the efficiency trade-off and execution time. As the mobile device was monitoring the packet loss by observing its runtime impact, it avoids the latency caused by network congestion.

Chapter 7

Conclusion

This PhD research was motivated by the primary issue of achieving energy-efficient mobile-cloud hybrid computing systems. The mobile applications, nowadays, are resource-rich and demand high computation power. When they execute on a mobile device, the hardware components of the device use the battery power. From a user point of view, a device never has enough battery power. In order to extend the battery life between charges, the computationally-intensive modules of mobile applications can be offloaded to the resource-rich cloud. Therefore, the computation power, which an application is supposed to consume from the battery of a mobile device, can be saved.

In this work, we have proposed a method to achieve energy efficient mobile-cloud hybrid applications created for Android-based (i.e., smartphones and tablets) and Linux-based (i.e., Raspberry Pi controlled robots) computing systems. We have considered the following two objectives.

1. Minimising battery power consumption: It is the total power consumption of an MC hybrid application, including computation and communication power, and power required for other components to operate such as LCD and memory. We measured this in joules.

2. Minimising network usage: It is bandwidth usage of the wireless network available to the mobile-devices (i.e., WIFI). We measured this in KBs.

The above two objectives are conflicting in nature because using computation offloading consumes power by transmitter chip on mobile devices. Therefore, minimising the two objectives creates a *efficiency trade-off*. We consider the effective partition of mobile applications as a multi-objective optimisation problem.

In this work, we have considered three testbed applications (developed using object-oriented programming paradigm): ImageEffects (Android-based), Mather (Android-based) and a foraging task (Python-based). We targetted two computing systems: 1) Android-based smartphones, and 2) Raspberry Pi controlled robots. In order to execute the applications on these computing systems, we first modularise the applications. It is the partitioning of code units into different offloadable modules. We represent these offloadable modules using class-level, method-level and hybrid level configurations. A configuration is a binary string that maps the offloadable modules to their machine where they will be executed during runtime of the application. We have presented our general purpose mobile-cloud hybrid application framework for the two computing systems. Our framework is based on the code-offloading technique to execute the offloadable modules of the MC hybrid applications remotely on the cloud using configurations. Based on the level of applications granularities, we consider a configuration one of three types: 1) class-level (coarse-grained), 2) method-level (fine-grained) and 3) hybrid - mix of coarse and fine grained. We create three levels of configuration sets for applications based on the configuration types.

In order to optimise the trade-off between power consumption and bandwidth usage, we created a workflow that uses multi-objective optimisation. The workflow automates the execution of the applications, and with an exhaustive search algorithm, it searches for efficient configurations using offline profiling. We used offline profiling to instrument the MC hybrid applications, which was measuring the power and network usage during runtime using the configurations. At the end of the offline profiling, the efficiency of all the configurations created for the MC hybrid applications was recorded.

To further improve the search for efficient configurations, we used statistical tests on the efficiency of configurations. We only obtained a final set of configurations that include: 1) Non-dominated configurations in the collapsible sets that are also statistically significant, and 2) other non-collapsible configurations. In the end, we find the Pareto-optimal configurations in the final set, which optimise the efficiency trade-off. Executing the MC hybrid applications with these configurations provide an alternative to achieve efficient mobile-cloud (MC) hybrid computing systems. From the results analysis, we also find out that increasing or decreasing the computation level of mobile applications has an impact on battery power consumption. Furthermore, the following observations were taken from the results.

1. Computation offloading is energy effective when computationally-intensive modules are offloaded to the cloud. The total power consumption, including computing and communication power, is low in this case.

2. Computation offloading is in-effective when less computationally-intensive modules are offloaded to the cloud. In this case, the total power consumption is lower when they are executed on mobile devices.

3. The mobile applications that have low computation level will result in the formation of horizontal type clusters of configurations that execute the same number of modules on the cloud when code offloading is used. On the other hand, mobile applications having high computation level will result in the formation of vertical type clusters of configurations.

4. The efficient configurations obtained for the MC hybrid applications had mixed granularity levels. This shows that the granularity level is important to consider during code offloading, to achieve energy-efficient mobile-cloud hybrid applications.

However, the exhaustive search algorithm used during offline profiling can take longer, depending on the total number of offloadable modules for an application. Since the number of configurations depends on the number of modules, the larger the configuration set (or modules), the more time the exhaustive search will take to find the efficient configurations. Estimated time for more than 20 modules would be more than a month. Therefore, we proposed a solution to scale up the method of achieving energy-efficient mobile-cloud hybrid applications. The scale-up strategy is composed of using evolutionary algorithms, such as NSGA-II, and an intelligent Two-Step search algorithm; which we introduced for searching efficient configurations for MC hybrid applications. They take less time to converge, albeit to approximate optimal solutions. From the experimental study, we obtained the following key facts.

1. For small-scale MC hybrid applications, the exhaustive search algorithm is appropriate to use for finding the Pareto-optimal configurations. In the Two-Step search algorithm, we used an exhaustive search to find Pareto-optimal class-level configurations (in the first step) because of a small number (4) of class-level offloadable modules for both ImageEffects and foraging task. The total number of configurations were 16 for which the exhaustive search algorithm took around 30 minutes to complete. However, in case of the offline profiling, the exhaustive search took more than a month to find the Pareto-optimal configurations for the total number of configurations ($32,400$) in all the three sets (class-level, method-level and hybrid-level) combined of the foraging task. According to our experiments, offloadable modules between 2 to 8 would lie the so-called small-scale MC hybrid applications. The class-level and method-level configurations of Mather are 2 and 6 respectively and can be considered a small-scale MC hybrid application.

2. For medium-scale MC hybrid applications, the exhaustive search algorithm will take a significant amount of time to complete and, therefore, would not be practical to use. In this case, evolutionary algorithms, such as NSGA-II, are appropriate to use, which can approximate the Pareto-front solutions in a reasonable amount of time. For the ImageEffects and Foraging task, where the number of class-level configurations for both is equal to 16, and the combined method-level and hybrid-level configurations are equal to $3,584$ and $32,384$, both the Two-Step and NSGA-II were feasible to use. They took about one week for the ImageEffects and two week time for the foraging task, for the ten independent runs to complete. According to our experiments, offloadable class-level modules less than 8 and method-level modules higher than 8 would lie the so-called medium-scale MC hybrid applications. ImageEffects and foraging task both can be considered medium scale MC hybrid applications. For both of these two MC hybrid applications, the Two-Step algorithm performed better than NSGA-II.

3. For large-scale MC hybrid applications, the Two-Step search algorithm would not be practical to use as the exhaustive search in step 1 would not complete in a feasible amount of time. For such applications, finding a scalable solution is an open

research question that we will leave for future work. According to our experiments, offloadable class-level and method-level modules higher than 8 would lie the so-called large-scale MC hybrid applications.

Since in the real world, the environment in which mobile devices operate changes randomly. To cope with the conditions such as weak signal strength, network link failure, network congestion, static offloading can cause latency and use more battery power (i.e., communication power for retransmission of dropped packets). In such cases a dynamic offloading decision is beneficial. Therefore, in addition to offline profiling, where static code offloading is used, our framework can also be used for online profiling as it is based on self-adaptive and self-aware decision mechanism. In the online profiling, the framework switches between the efficient configurations, based on a change in the environment and change within the system itself, to minimise battery power consumption, network usage and improve the performance of applications by avoiding network latency.

We were able to simulate the change in the operating environment, in a lab-based controlled setup, by using a workflow for online profiling of the MC hybrid applications. When the signal level was good, the framework would switch to a configuration that executes a maximum number of modules on the cloud. In case of a weak signal level, the framework would switch to a configuration that executes all modules on the device. The analysis of our experimental study provided some key facts regarding online optimisation. They are enumerated as follow.

1. By simulating a scenario in which a mobile device was moving into the network coverage area experiencing a change in signal level. We observed that using the self-adaptive and self-aware decisions performed better than using static offloading and no offloading, in terms of completion time and optimising the efficiency trade-off.

2. By simulating a scenario in which a mobile device was moving into the network coverage area experiencing: 1) a change in signal level, and 2) network latency caused due to congestion in the network. We observed that the self-aware decision

mechanism performed better than the self-adaptive, static offloading, and no offloading - in terms of optimising the efficiency trade-off and execution time. As the mobile device was monitoring the packet loss by observing its runtime impact, it avoids the latency caused by network congestion.

The proposed approach to achieve energy-efficient HMC applications can be used by developers of applications. Using the offline profiling they will be able to find the Pareto-optimal configurations of their application. They can then use those configurations to deploy with the HMC application. By using the framework, the self-adaptive and self-aware decision mechanisms will chose the configurations during runtime of the application.

In the end, we hope that the proposed method will provide researchers and developers insights to developing energy-efficient mobile-cloud hybrid alternative to Android-based and Python-based computing systems.

7.0.1 Future directions

To validate the idea of energy-efficient, scalable, self-adaptive and self-aware mobile-cloud hybrid computing systems, we presented our technique and discussed the experimental results. However, this work can be extended in some areas. We discuss some future direction as follow.

In this work, we have considered applications developed using Object Oriented Programming paradigm for mobile devices. We modularise their code based on classes and methods. We create configurations of applications which are coarse-grained (class-level) or fine-grained (method-level). However, there are other types of programming paradigms that are also used to develop application software, i.e., functional programming and procedural programming. Moreover, cross-platform mobile applications are developed using HTML, Javascript and XML etc, where the concept of OOP (i.e., classes) is not followed. This is one area that is strongly recommended to be explored in the future.

Secondly, in this work, as we have mentioned in Chapter 3, the programmers of an HMC application annotate the code manually during the development time. This is done to identify offloadable modules in the code. A converter application is then used to analyse the source code and inject the offloading interface for the modules that were annotated. In future work, static analysis of the applications can be used to automatically find which modules are suitable for code offloading. The computationally-intensive modules can be identified by techniques such as the number of lines of code in a module, data types used in a module, the number of times a loop will execute etc.

Thirdly, we have used multi-objective optimisation to minimise two objectives, battery power consumption and network usage, using offline profiling. The future work could extend the multi-objective optimisation to more than two objectives. For example, the objectives of minimising delay and maximising application's performance could be added to offline profiling. Moreover, the experimental work using offline profiling can be extended to other underlying technologies, e.g., mobile-edge and mobile-fog. Currently, we have only considered mobile-cloud as the underlying technology.

Fourthly, the Two-Step search algorithm, which we proposed in this work, converges in a feasible amount of time for small to medium size applications. For large scale applications, where the offloadable class-level and method-level modules are higher than 8, it would not be practical to use the Two-Step algorithm. This is because the exhaustive search in step 1 would not complete in a feasible amount of time. Instead, a fast search algorithm such as NSGA-II can be used in future work.

Fifthly, the self-adaptive and self-aware decision mechanisms only take into consideration the signal level and packet loss as the changing factors. Other factors such as CPU usage can be added in the future work. For example, when the Operating System is using too much processing power for system-related tasks, the HMC application will switch to a configuration that offloads the code to the cloud. Furthermore, in future work, a formal method/algorithm can be implemented for the self-adaptive and self-aware approaches. Also, the framework can be evaluated with other contextual factors such as resource usage, code type, cloud-side context and using a non-WIFI network (e.g., LoRa/LoRaWAN, NB-IoT, SigFox)

Sixthly, the proposed HMC framework is only limited to use by the developer or service provider to develop applications. In future work, a client-server infrastructure can be developed so that everyone can directly upload any application to the server. The server will be using tools to optimise and scale the application to HMC application using the framework. The user can then use the HMC version of the application, which would provide an energy-efficient alternative.

Bibliography

[1] Amazon simple storage service (amazon s3) is an object storage service. https://aws.amazon.com/s3/.

[2] Amazon web services (aws). https://aws.amazon.com/.

[3] A completely new rmi implementation to replace native java rmi. http://lipermi.sourceforge.net/.

[4] Dropbox is a cloud-based file storing service. https://www.dropbox.com/.

[5] Firebase is a realtime database. https://firebase.google.com/.

[6] Google cloud app engine. https://cloud.google.com/appengine/.

[7] Google photos is a cloud-based photos/video sharing and storing service. https://photos.google.com/.

[8] Microsot azure provides cloud comtputing service. https://azure.microsoft.com/.

[9] Netem provides, network emulation, functionality for testing protocols by emulating the properties of wide area networks. https://wiki.linuxfoundation.org/networking/netem.

[10] Rackspace cloud computing services. https://www.rackspace.com/.

[11] C2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401 – 413, 2014.

[12] A. Abbas and S. U. Khan. A review on the state-of-the-art privacy-preserving approaches in the e-health clouds. *IEEE Journal of Biomedical and Health Informatics*, 18(4):1431–1441, July 2014.

[13] E. Ahmed, A. Gani, M. Sookhak, S. H. A. Hamid, and F. Xia. Application optimization in mobile cloud computing. *J. Netw. Comput. Appl.*, 52(C):52–68, June 2015.

[14] E. Ahmed and M. H. Rehmani. Mobile edge computing: Opportunities, solutions, and challenges. *Future Generation Computer Systems*, 70:59 – 63, 2017.

[15] A. Akbar and P. R. Lewis. Towards the optimization of power and bandwidth consumption in mobile-cloud hybrid applications. In *Proceedings of the 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 213–218, May 2017.

[16] A. Akbar and P. R. Lewis. The importance of granularity in multiobjective optimization of mobile cloud hybrid applications. *Transactions on Emerging Telecommunications Technologies*, 10 2018.

[17] A. Akbar and P. R. Lewis. Self-adaptive and self-aware mobile-cloud hybrid robotics. In *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, pages 262–267, Oct 2018.

[18] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak. Energy cost models of smartphones for task offloading to the cloud. *IEEE Transactions on Emerging Topics in Computing*, 3(3):384–398, 2015.

[19] J. H. Andrews, T. Menzies, and F. C. Li. Genetic algorithms for randomized unit testing. *Ieee transactions on software engineering*, 37(1):80–94, 2011.

[20] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[21] J. S. Arora. *Introduction to optimum design*. Elsevier, 2004.

[22] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit. Davinci: A cloud computing framework for service robots. In *2010 IEEE International Conference on Robotics and Automation*, pages 3084–3089, May 2010.

[23] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[24] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler. Theory of the hypervolume indicator: optimal μ-distributions and the choice of the reference point. In *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, pages 87–102. ACM, 2009.

[25] R. Aversa, B. Di Martino, M. Rak, and S. Venticinque. Cloud agency: A mobile agent based cloud system. In *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 132–137, Feb 2010.

[26] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, pages 87–92, New York, NY, USA, 2002. ACM.

[27] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani. Data center network virtualization: A survey. *IEEE Communications Surveys & Tutorials*, 15(2):909–928, 2013.

[28] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, pages 13–16. ACM, 2012.

[29] L. Bradstreet. *The hypervolume indicator for multi-objective optimisation: calculation and use*. University of Western Australia, 2011.

[30] G. Brown. Mobile edge computing use cases and deployment options. *Juniper White Paper*, pages 1–10, 2016.

[31] R. Buyya. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 1–, Washington, DC, USA, 2009. IEEE Computer Society.

[32] E. Çalışır, G. I. Alptekin, and B. A. Özgövde. A code offloading framework for mobile cloud computing: Icemobile.

[33] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli. Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):795–809, 2004.

[34] Y. Chen, Z. Du, and M. García-Acosta. Robot as a service in cloud computing. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, pages 151–158. IEEE, 2010.

[35] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[36] S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan. How close is close enough? understanding the role of cloudlets in supporting display appropriation by mobile users. In *2012 IEEE International Conference on Pervasive Computing and Communications*, pages 122–127. IEEE, 2012.

[37] C. A. Coello. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys (CSUR)*, 32(2):109–143, 2000.

[38] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[39] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[40] F. F. de Vega, F. Chávez, J. Díaz, J. A. García, P. A. Castillo, J. J. Merelo, and C. Cotta. A cross-platform assessment of energy consumption in evolutionary algorithms. In J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, editors, *Parallel Problem Solving from Nature – PPSN XIV*, pages 548–557, Cham, 2016. Springer International Publishing.

[41] K. Deb. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*, pages 3–34. Springer, 2011.

[42] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, pages 849–858, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[43] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya. Computation offloading for service workflow in mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(12):3317–3329, Dec 2015.

[44] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13, 12 2013.

[45] D. Doval, S. Mancoridis, and B. S. Mitchell. Automatic clustering of software systems using a genetic algorithm. In *Software Technology and Engineering Practice, 1999. STEP'99. Proceedings*, pages 73–81. IEEE, 1999.

[46] N. Dutt, A. Jantsch, and S. Sarma. Toward smart embedded systems: A self-aware system-on-chip (soc) perspective. *ACM Trans. Embed. Comput. Syst.*, feb 2016.

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

[48] F. Ferrucci, M. Harman, J. Ren, and F. Sarro. Not going to take this anymore: Multi-objective overtime planning for software engineering projects. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 462–471, Piscataway, NJ, USA, 2013. IEEE Press.

[49] F.Kaup, P.Gottschling, and D.Hausheer. Powerpi: Measuring and modeling the power consumption of the raspberry pi. In *Proceedings of the 39th Annual IEEE Conference on Local Computer Networks*, Sept 2014.

[50] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya. Mobile code offloading: from concept to practice and beyond. *IEEE Communications Magazine*, 53(3):80–88, 2015.

[51] H. Flores and S. Srirama. Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning. In *Proceeding of the 4th ACM MobiSys workshop on Mobile cloud computing and services*, pages 9–16. ACM, 2013.

[52] H. Flores, S. N. Srirama, and C. Paniagua. A generic middleware framework for handling process intensive hybrid cloud services from mobiles. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '11, pages 87–94, New York, NY, USA, 2011. ACM.

[53] H. Flores, S. N. Srirama, and C. Paniagua. Towards mobile cloud applications: Offloading resource-intensive tasks to hybrid clouds. *International Journal of Pervasive Computing and Communications*, 8(4):344–367, 2012.

[54] C. M. Fonseca, J. D. Knowles, L. Thiele, and E. Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. In *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 216, page 240, 2005.

[55] R. Friedman and N. Hauser. Coara: Code offloading on android with aspectj. *arXiv preprint arXiv:1604.00641*, 2016.

[56] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[57] A. Ghosh, R. Ratasuk, B. Mondal, N. Mangalvedhe, and T. Thomas. Lte-advanced: next-generation wireless broadband technology [invited paper]. *IEEE Wireless Communications*, 17(3):10–22, June 2010.

[58] A. Gohil, H. Modi, and S. K. Patel. 5g technology of mobile communication: A survey. In *2013 international conference on intelligent systems and signal processing (ISSP)*, pages 288–292. IEEE, 2013.

[59] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[60] E. I. S. Group. Mobile-edge computing. *Introductory Tech. White Paper*, page 1–36, Sep 2014.

[61] X. Gu, A. Messer, I. Greenberg, D. Milojicic, and K. Nahrstedt. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing*, 3(3):66–73, July 2004.

[62] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading inference for delivering applications in pervasive computing environments. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003).*, pages 107–114. IEEE, 2003.

[63] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li. Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing. *IEEE Transactions on Mobile Computing*, 18(2):319–333, 2019.

[64] S. Guo, B. Xiao, Y. Yang, and Y. Yang. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[65] M. Harman and B. F. Jones". "search-based software engineering ". *Information and Software Technology*, 43(14):833 – 839, 2001.

[66] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo. Empirical software engineering and verification. chapter Search Based Software Engineering: Techniques, Taxonomy, Tutorial, pages 1–59. Springer-Verlag, 2012.

[67] W. Haynes. *Wilcoxon Rank Sum Test*, pages 2354–2355. Springer New York, New York, NY, 2013.

[68] J. Heinerman, A. Zonta, E. Haasdijk, and A. E. Eiben. On-line evolution of foraging behaviour in a population of real robots. In G. Squillero and P. Burelli, editors,

Applications of Evolutionary Computation, pages 198–212, Cham, 2016. Springer International Publishing.

[69] R. V. Hogg and E. A. Tanis. *Probability and statistical inference*. Pearson Educational International, 2009.

[70] G. Hu, W. P. Tay, and Y. Wen. Cloud robotics: architecture, challenges and applications. *IEEE network*, 26(3):21–28, 2012.

[71] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[72] D. Huang et al. Mobile cloud computing. *IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter*, 6(10):27–31, 2011.

[73] G. Huerta-Canepa and D. Lee. An adaptable application offloading scheme based on application behavior. In *22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008)*, pages 387–392. IEEE, 2008.

[74] C.-L. Hwang and A. S. M. Masud. *Multiple objective decision making—methods and applications: a state-of-the-art survey*, volume 164. Springer Science & Business Media, 2012.

[75] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on automation science and engineering*, 12(2):398–409, 2015.

[76] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: A computation offloading framework for smartphones. In M. Gris and G. Yang, editors, *Mobile Computing, Applications, and Services*, volume 76 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 59–79. Springer Berlin Heidelberg, 2012.

[77] A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan. A survey of mobile cloud computing application models. *Communications Surveys & Tutorials, IEEE*, 16(1):393–413, 2014.

[78] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006.

[79] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Unleashing the power of mobile cloud computing using thinkair. *CoRR*, abs/1105.3232, 2011.

[80] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Infocom, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[81] S. Kounev, P. Lewis, K. L. Bellman, N. Bencomo, J. Camara, A. Diaconescu, L. Esterle, K. Geihs, H. Giese, S. Götz, P. Inverardi, J. O. Kephart, and A. Zisman. *The Notion of Self-aware Computing*, pages 3–16. Springer, 2017.

[82] J. KUFFNER. Cloud-enabled humanoid robots. *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on, Nashville TN, United States, Dec.*, 2010.

[83] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[84] K. Kumar and Y. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, April 2010.

[85] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao. A survey of self-awareness and its application in computing systems. In *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 102–107, Oct 2011.

[86] P. R. Lewis, M. Platzner, B. Rinner, J. Torresen, and X. Yao, editors. *Self-Aware Computing Systems: An Engineering Approach*. Springer, 2016.

[87] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: A partition scheme. In *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '01, pages 238–246, New York, NY, USA, 2001. ACM.

[88] X. Lin, Y. Wang, Q. Xie, and M. Pedram. Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. *IEEE Transactions on Services Computing*, 8(2):175–186, March 2015.

[89] M. Linares-Vásquez, C. Bernal-Cárdenas, G. Bavota, R. Oliveto, M. Di Penta, and D. Poshyvanyk. Gemma: Multi-objective optimization of energy consumption of guis in android apps. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, pages 11–14, Piscataway, NJ, USA, 2017. IEEE Press.

[90] B. Liu, Y. Chen, E. Blasch, K. Pham, D. Shen, and G. Chen. A holistic cloud-enabled robotics system for real-time video tracking application. In *Future Information Technology*, pages 455–468. Springer, 2014.

[91] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi. Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, 48:99 – 117, 2015.

[92] M. López-Ibáñez, T. Stützle, and L. Paquete. Graphical tools for the analysis of bi-objective optimization algorithms:[workshop on theoretical aspects of evolutionary multiobjective optimization]. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 1959–1962. ACM, 2010.

[93] J. Lorch and A. Smith. Software strategies for portable computer energy management. *IEEE Personal Communications*, 5(3):60–73, 1998.

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

[95] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys Tutorials*, 19(4):2322–2358, Fourthquarter 2017.

[96] R. Marler and J. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, Apr 2004.

[97] A. Messer, I. Greenberg, D. Miljicic, P. Bernadat, and G. Fu. Method and system for offloading execution and resources for resource-constrained networked devices, Jan. 24 2006. US Patent 6,990,662.

[98] A. P. Miettinen and J. K. Nurminen. Energy efficiency of mobile clients in cloud computing. In *HotCloud*, 2010.

[99] L. L. Minku and X. Yao. Software effort estimation as a multiobjective learning problem. *ACM Trans. Softw. Eng. Methodol.*, 22(4):35:1–35:32, Oct. 2013.

[100] F. A. Nakahara and D. M. Beder. A context-aware and self-adaptive offloading decision support model for mobile cloud computing system. *Journal of Ambient Intelligence and Humanized Computing*, Apr 2018.

[101] N. Z. Naqvi, J. Devlieghere, D. Preuveneers, and Y. Berbers. Mascot: Self-adaptive opportunistic offloading for cloud-enabled smart mobile applications with probabilistic graphical models at runtime. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan 2016.

[102] G. Oliveira and V. Isler. View planning for cloud-based active object recognition. *Department of Computer Science, University of Minnesota, Tech. Rep*, 2013.

[103] S. K. Pal, S. Bandyopadhyay, and C. Murthy. Genetic algorithms for generation of class boundaries. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(6):816–828, 1998.

[104] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, 17(02):223–255, 2008.

[105] K. Praditwong, M. Harman, and X. Yao. Software module clustering as a multi-objective search problem. *IEEE Trans. Softw. Eng.*, 37(2):264–282, Mar. 2011.

[106] J. S. Preden, K. Tammemäe, A. Jantsch, M. Leier, A. Riid, and E. Calis. The benefits of self-awareness and attention in fog and mist computing. *Computer*, 48(7):37–45, 2015.

[107] A. Rahman, J. Jin, A. Cricenti, A. Rahman, and D. Yuan. A cloud robotics framework of optimal task offloading for smart city applications. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, Dec 2016.

[108] M. Rahman, J. Gao, and W. Tsai. Energy saving in mobile cloud computing*. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pages 285–291, March 2013.

[109] B. S. P. Reddy and C. S. P. Rao. A hybrid multi-objective ga for simultaneous scheduling of machines and agvs in fms. *The International Journal of Advanced Manufacturing Technology*, 31(5):602–613, Dec 2006.

[110] E. Rodriguez-Tello and J. Torres-Jimenez. Era: An algorithm for reducing the epistasis of sat problems. In *Genetic and Evolutionary Computation—GECCO 2003*, pages 1283–1294. Springer, 2003.

[111] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. Saving portable computer battery power through remote process execution. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):19–26, Jan. 1998.

[112] O. Saha and P. Dasgupta. A comprehensive survey of recent trends in cloud robotics architectures and applications. *Robotics*, 7(3):47, 2018.

[113] M. Satyanarayanan. Fundamental challenges in mobile computing. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 1996.

[114] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang. Cyber-physical systems: A new frontier. In *2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*, pages 1–9. IEEE, 2008.

[115] M. Shiraz, A. Gani, A. Shamim, S. Khan, and R. W. Ahmad. Energy efficient computational offloading framework for mobile cloud computing. *Journal of Grid Computing*, 13(1):1–18, 2015.

[116] B. Siciliano and O. Khatib. *Springer handbook of robotics*. Springer, 2016.

[117] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[118] P. C. . F. H. Srirama, S.N. 2012.

[119] K. O. Stanley. *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 2004.

[120] I. Stojmenovic and S. Wen. The fog computing paradigm: Scenarios and security issues. In *2014 Federated Conference on Computer Science and Information Systems*, Sept 2014.

[121] K. C. Tan, T. H. Lee, and E. F. Khor. Evolutionary algorithms for multi-objective optimization: performance assessments and comparisons. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, volume 2, pages 979–986 vol. 2, May 2001.

[122] E. Tilevich and Y. Smaragdakis. J-orchestra: Automatic java application partitioning. In *Proceedings of the 16th European Conference on Object-Oriented Programming*, pages 178–204. Springer-Verlag, 2002.

[123] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili. Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges. *arXiv preprint arXiv:1612.03184*, 2016.

[124] L. Turnbull and B. Samanta. Cloud robotics: Formation control of a multi robot system utilizing cloud infrastructure. In *2013 Proceedings of IEEE Southeastcon*, pages 1–4. IEEE, 2013.

[125] A. u. R. Khan, M. Othman, F. Xia, and A. N. Khan. Context-aware mobile cloud computing and its challenges. *IEEE Cloud Computing*, May 2015.

[126] A. Valcarce, T. Rasheed, K. Gomez, S. Kandeepan, L. Reynaud, R. Hermenier, A. Munari, M. Mohorcic, M. Smolnikar, and I. Bucaille. Airborne base stations for emergency and temporary events. In *International conference on personal satellite services*, pages 13–25. Springer, 2013.

[127] B. Vic and L. Toby. *Outliers in Statistical Data, (3rd ed.)*. Wiley, 1994.

[128] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos. Cloud robotics: Current status and open issues. *IEEE Access*, 4:2797–2807, 2016.

[129] Q. Wang and R. Deters. Soa's last mile-connecting smartphones to the service cloud. In *2009 IEEE International Conference on Cloud Computing*, pages 80–87, Sep. 2009.

[130] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5:6757–6779, 2017.

[131] Y. Wen, W. Zhang, and H. Luo. Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In *2012 Proceedings Ieee Infocom*, pages 2716–2720. IEEE, 2012.

[132] H. Wu, Q. Wang, and K. Wolter. Methods of cloud-path selection for offloading in mobile cloud computing systems. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 443–448. IEEE, 2012.

[133] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. 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, New York, NY, USA, 2010. ACM.

[134] W. Zhang, A. J. Berre, D. Roman, and H. A. Huru. Migrating legacy applications to the service cloud. In *14th Conference companion on Object Oriented Programming Systems Languages and Applications (OOPSLA 2009)*, pages 59–68, 2009.

[135] Z. Zhang, K. Long, J. Wang, and F. Dressler. On swarm intelligence inspired self-organized networking: Its bionic mechanisms, designing principles and optimization approaches. *IEEE Communications Surveys Tutorials*, 16(1):513–537, First 2014.

[136] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32 – 49, 2011.

[137] J. Zhu, X. Fang, Z. Guo, M. H. Niu, F. Cao, S. Yue, and Q. Y. Liu. Ibm cloud computing powering a smarter planet. In *IEEE International Conference on Cloud Computing*, pages 621–625. Springer, 2009.

[138] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, Nov 1999.

Glossary

battery-power consumption The power usage by a mobile device to execute an application. 17

computationally-intensive tasks Those tasks performed by a mobile device during execution of applications that require do complex computation, therefore, consumes high battery power.. 17

computation offloading Computation or code offloading is a technique to execute part of mobile applications on a remote server or the cloud.. 17

configurations Throughout this thesis, a configuration would refer to a binary string that represent the offloadable modules of an HMC application.. 19

GPS Global Positioning System. 15

HMC Throughout the thesis, HMC would refers to hybrid mobile-cloud application(s). 2, 16

hybrid-level configurations A hybrid-level configuration or sometimes refer as hybrid configuration is which can run an HMC application in both fine and coarse grained level of granularity.. 60

latency The delay caused during transmission of data in a network. In other words, the round trip time from the browser to the server.. 17

MCC Throughout the thesis, MCC would refers to mobile-cloud computing paradigm. 16

MOO Throughout the thesis, MOO would refers to Multi-Objective Optimisation. 18

multi-objective optimisation Optimising two or more objectives.. 18, 20

network usage It is the amount of data sent and received during one execution of an application.. 17

Pareto-optimal solutions Those solutions that are not dominated by any other solutions in the solution set. They provide a compromise between the objectives.. 18

resource-hungry Limited amount of resources available to an application or a mobile device. Other similar terms like resource-constrained or resource-limited are also used.. 15