THESIS

AN INTELLIGENT, MOBILE NETWORK AWARE MIDDLEWARE FRAMEWORK FOR ENERGY EFFICIENT OFFLOADING IN SMARTPHONES

Submitted by

Aditya Dilip Khune

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2017

Master's Committee:

Advisor: Sudeep Pasricha

Anura P. Jayasumana
Bob Gesumaria

ABSTRACT

AN INTELLIGENT, MOBILE NETWORK AWARE MIDDLEWARE FRAMEWORK FOR ENERGY EFFICIENT OFFLOADING IN SMARTPHONES

Offloading mobile computations is an innovative technique that is being explored by researchers for reducing energy consumption in mobile devices and for achieving better application response time. Offloading refers to the act of transferring computations from a mobile device to servers in the cloud. There are many challenges in this domain that are not dealt with effectively yet, and thus offloading is far from being adopted in the design of current mobile architectures. We believe that there is a need to verify the effectiveness of computation offloading in terms of both response time and energy consumption, to highlight its potential in real smartphone applications. The effect of varying network technologies such as 3G, 4G, and Wi-Fi on the performance of offloading systems is also a major concern that needs to be addressed. In this thesis, we study the behavior of a set of real smartphone applications, in both local and offload processing modes. Our experiments identify the advantages and disadvantages of offloading for various mobile networks. Further, we propose a middleware framework that uses Reinforcement Learning to make reward-based offloading decisions effectively. Our framework allows a smartphone to consider suitable contextual information to determine when it makes sense to offload, and to select between available networks (3G, 4G, or Wi-Fi) when offloading mode is active. We tested our framework in both simulated and real environments, across various applications, to demonstrate how energy consumption can be minimized in mobile systems that are capable of supporting offloading.

ACKNOWLEDGEMENTS

I would like to express my special appreciation and thanks to my advisor Prof. Sudeep Pasricha, whose guidance, support, and encouragement made the research presented in this thesis possible. Thank you, Prof. Pasricha, for being patient, and giving me the time to explore.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Anura Jayasumana and Prof. Bob Gesumaria for serving as my committee members. I would also like to thank my colleagues in the Embedded Systems and High Performance Computing (EPiC) lab for their advice and unwavering support. I am grateful to Hrushikesh Kulkarni, who as a good friend was always willing to help and give his best suggestions.

Many thanks to everyone at the Bhagavad-Gita group at Fort Collins and to the spiritual teacher in the class Sanatan Priya Prabhu who has supported me through good and bad times.

Last but not least, I would like to thank my family, my loving parents Dilip Khune, Geeta Khune, and my sister Apurva for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

DEDICATION

To my loving parents and sister

Without their support, understanding, encouragement, and love this work would not have been possible.

TABLE OF CONTENTS

ABSTRACT ii
ACKNOWLEDGEMENTS iii
DEDICATION iv
LIST OF TABLES vii
LIST OF FIGURES viii
CHAPTER 1 INTRODUCTION 1
1.1 Mobile cloud computing 2
1.2 Offloading smartphone applications 3
1.3 Outline 3
CHAPTER 2 PRIOR WORK 5
CHAPTER 3 CHALLENGES WITH OFFLOADING 9
CHAPTER 4 OFFLOADING PERFORMANCE OF MOBILE APPLICATIONS 11
4.1 Experimental Setup 12
4.2 Experimental Results 14
4.2.1 Matrix operation app 14
4.2.2 Internet browser app 16
4.2.3 Zipper app 17
4.2.4 Voice recognition and translation app 19
4.2.4 Torrents app 20
4.3 Summary of findings 21
CHAPTER 5 ADAPTIVE OFFLOADING 24
CHAPTER 6 MIDDLEWARE FRAMEWORK FOR EFFICENT OFFLOADING OF MOBILE APPLICATIONS 27
6.1 RL algorithm to generate Q function 30
CHAPTER 7 EXPERIMENTAL RESULTS 34
CHAPTER 8 CONCLUSION AND FUTURE WORK 38
8.1 Conclusion 38
8.2 Future Work 38
REFERENCES 40
APENDIX A SOURCE CODE 43
A. RL1_Mainoffloadingappactivity.java 43
B. Fuzzylogicdisplay.java 46

C. RL2_Reinforcement_strategy.py 47
D. RL3_RLstrategy.java 51
E. Mobile-AWS cloud interaction -1 54
F. Mobile-AWS cloud interaction - 2 55
G. LDAStrategy.java 55
ABBREVATIONS 58

LIST OF TABLES

Table 4.1 Accuracy of voice recognition and translation app for local vs. cloud processing 20

Table 6.1 Penalty values in RL algorithm 32

LIST OF FIGURES

Fig. 1.1 Number of smartphone users [1] 1

Fig. 1.2 Mobile Cloud Computing [38] 2

Fig. 2.1 Offloading communication vs computation [4] 5

Fig. 2.2 Offloading architecture [5] 6

Fig. 4.1 Monsoon power monitor setup 12

Fig. 4.2 Average battery consumption and average response time on a mobile device for a matrix operation with varying matrix sizes 15

Fig. 4.3 Average battery consumption and response time on a mobile device for an internet browsing session with varying data sizes 17

Fig. 4.4 Average battery consumption and response time on a mobile device for zipping/compressing files of varying sizes 18

Fig. 4.5 Average battery consumption on a mobile device for voice recognition and translation operations 19

Fig. 4.6 Average battery consumption and response time on a mobile device for torrent file download operations 21

Fig. 5.1 Offloading decision engine mechanism 24

Fig. 5.2 Fuzzy decision engine mechanism [10] 25

Fig. 6.1 Reinforcement Learning (RL) based middleware framework for efficient application offloading to cloud 27

Fig. 6.2 Q-learning flow and Q-table 28

Fig. 6.3 Decision making using Q-table (vector of key value pairs) 33

Fig. 7.1 Average battery consumption and response time of Matrix operations app with learning methods 35

Fig. 7.2 Average battery consumption and response time of Zipper app with learning methods 36

Fig. 7.3 Average battery consumption and response time of Torrent app with learning methods 37

Fig. 7.4 Prediction accuracy of learning methods 37

CHAPTER 1

INTRODUCTION

Faster wireless network speeds and rapid innovations in mobile technologies have changed the way we use our computers. It is estimated that 207.2 million people in the U.S. own a smartphone today while the number of smartphone users worldwide is estimated to be more than two billion [1]. Figure 1.1 shows the projected number of smartphone users until the year 2019. These mobile devices are not only used for making voice calls but are also efficiently able to run complex mobile applications that interact with the Internet. The volume of data being accessed and processed by smartphones, and the sophistication of mobile applications is rapidly increasing over time. However, the rapid evolution in hardware and software capabilities of mobile devices has not been paralleled by a similar advance in battery technology [2]. As expected, high end mobile

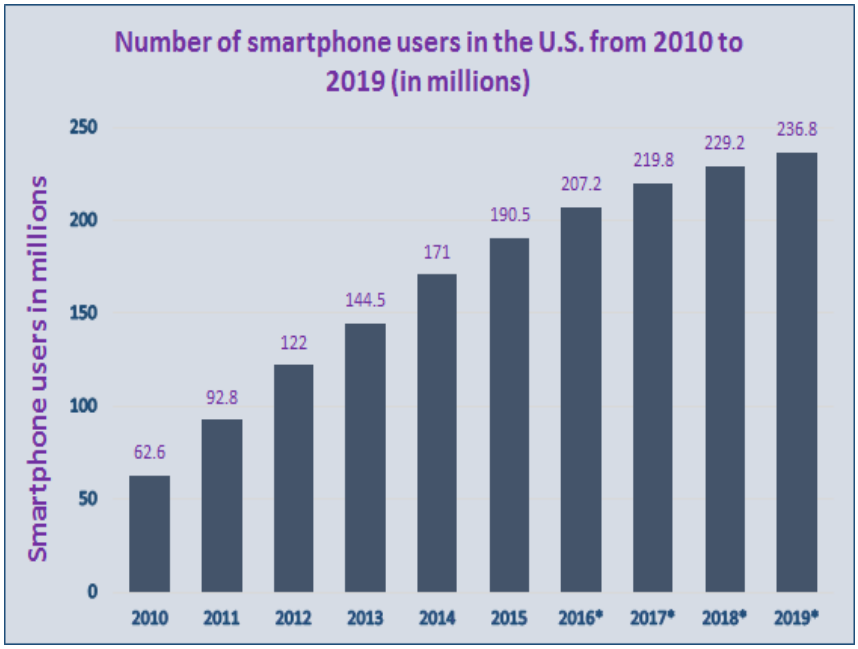

Fig. 1.1 Number of smartphone users [1]

applications increase the burden on the battery life of smartphones. For example, it has been shown

that a GPS based smartphone app can drain a mobile phone's battery completely within seven hours [3].

1.1 Mobile cloud computing

Cloud computing has drawn the attention of mobile technologies due to the increasing demand of applications for processing power, storage, and energy. Cloud computing promises availability of virtually infinite resources and it operates with utility computing model, where consumers pay based on their usage. Today, a number of applications are already using mobile cloud computing, for example social networking apps, location based services, sensor based health-care apps, gaming apps etc.

Fig. 1.2 Mobile Cloud Computing [38]

1.2 Offloading smartphone applications

A promising solution that is being considered to support high end mobile applications is to offload mobile computations to the cloud [6-11]. Offloading is an opportunistic process that relies on cloud servers to execute the functionality of an application that typically runs on a mobile device. The terms "cyber foraging" and "surrogate computing" are also sometimes used to describe such computation offloading. Such computation offloading is being considered today as a means to save energy and increase the responsiveness of mobile applications. The potential of computation offloading lies in the ability to sustain power hungry applications by releasing the energy consuming resources of the smartphone from intensive processing.

In this thesis, we study the behavior of a set of popular smartphone applications, in both local and offload processing modes. This study is crucial to examine the pros and cons of offloading when using various wireless networks. Further, we propose a novel middleware framework that uses a reward-based machine-learning technique called Reinforcement Learning (RL) to make offloading decisions effectively. The proposed framework considers various types of information on the mobile device, such as network type, network bandwidth, user-context, etc., to decide when to offload in order to minimize energy consumption. Our strategy also helps to select between available networks (3G, 4G, or Wi-Fi) when offloading mode is active. Our experiments with real applications on a smartphone highlight the potential of our framework to minimize energy in mobile devices that are capable of supporting offloading.

1.3 Outline

The rest of the thesis is organized as follows. Chapter 2 gives an overview of related prior works in offloading. In our literature review, we discuss important findings and shortcomings of related

publications and give an idea of what our framework can offer in order to improve offloading system in mobile devices. In chapter 3, we briefly discuss major challenges that are faced in building efficient offloading system. Chapter 4 presents a comprehensive application oriented study of offloading using available networks such as 3G, 4G and, Wi-Fi. We selected a set of five popular smartphone applications for our experiments; we present our findings based on those results in the last subsection of chapter 4. In chapter 5, we discuss the need for an adaptive offloading approach for making an effective offloading decision. In chapter 6, we present a middleware framework for efficient offloading of mobile applications. Our proposed framework is based on machine learning technique called as reinforcement learning. We conducted a series of experiments in chapter 7 to evaluate the efficacy of our proposed framework as compared to a related prior work. Chapter 6 concludes the thesis with a summary. The appendix offers the source code of the strategies presented in our thesis.

CHAPTER 2

PRIOR WORK

In this Chapter, we briefly review the related literature in the field of mobile computation offloading, illustrate the difference between other work and ours, and finally suggest some improvements in the offloading techniques presented in the related literature.

Kumar et al. [4] presented a mathematical analysis of offloading. Broadly, the energy saved by computation offloading depends on the amount of computation to be performed (C), the amount of data to be transmitted (D) and the wireless network bandwidth (B). If (D/C) is low, then it was claimed that offloading can save energy. Figure 2.1 briefly summarizes findings of the mathematical analysis; offloading is beneficial for compute intensive applications which use minimal communication with the cloud servers. In our experiments presented in chapter 4 of this thesis, we study other possible factors that affect offloading besides computations and communication.

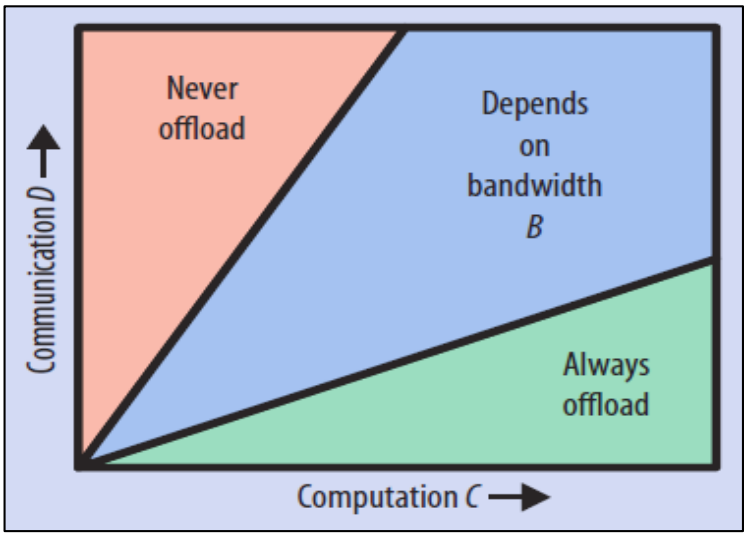

Fig. 2.1 Offloading communication vs computation [4]

Many prior research efforts have proposed strategies to empower mobile devices with offloading capabilities. Flores et al. [10] proposed a fuzzy decision engine for code offloading. The mobile device uses the decision engine based on fuzzy logic to combine various factors and decide when to offload. Problem with fuzzy decision engine approach is that the app developers will have to customize the decision engines depending upon which part of the world the device lies to account for variations in the different technologies around the world, for example in India, a country with limited broadband infrastructure, 2G remains in active use, while the U.S. and Mexico lean heavily on Wi-Fi connections. Our framework proposed in this thesis considers many more factors than these works, such as such as network type, data size, and degree of computations when making decisions about offloading.

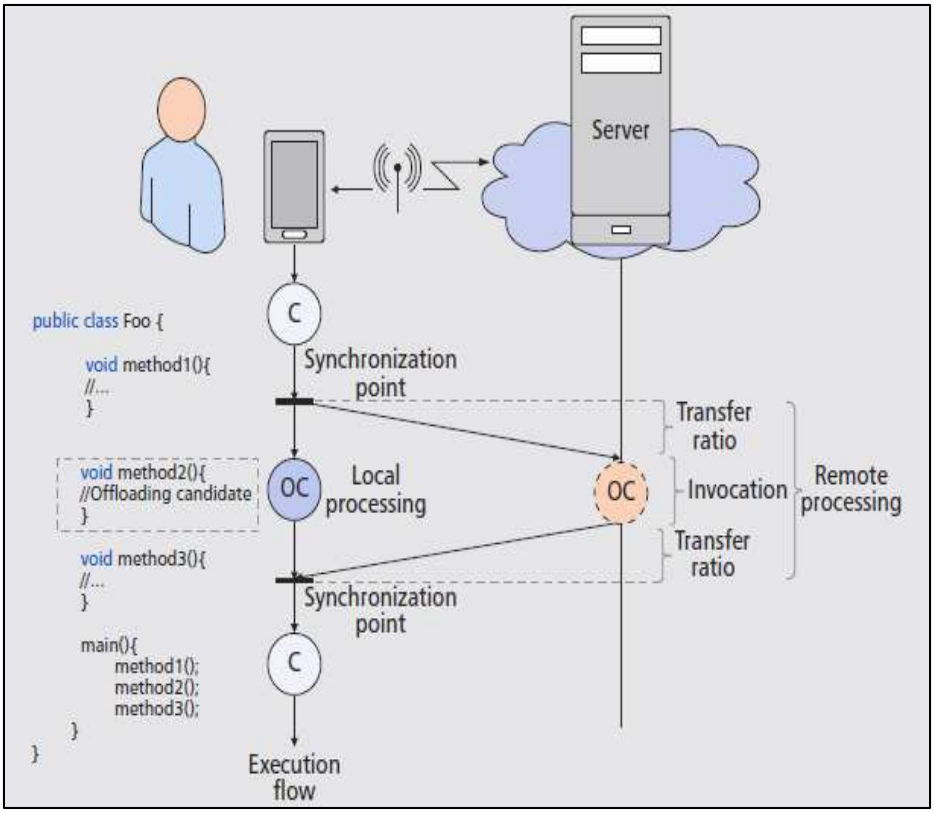

Fig. 2.2 Offloading architecture [5]

Offloading relies on remote servers to execute code delegated by a mobile device. In this process, the mobile is granted the local decision logic to identify resource-intensive portions of code, such that the mobile can estimate where the execution of code will require less computational effort (remote or local), which leads the device to save energy. Figure 2.2 shows an offloading architecture that is presented by Flores et al. [5]. This framework monitors multiple parameters of a smartphone, such as available bandwidth, data size to transmit, and energy to execute the code. The system considers these parameters to know when to offload, what to offload (portions of code: Method or Thread). The architecture consists of two parts, a client and a server. The client is composed of a code profiler, system profilers, and a decision engine. The server contains the surrogate platform to invoke and execute code. Code partitioning is a complex process that requires selection of the application code to be offloaded referred as offloading candidate (OC) in the figure.

Chun et al. [7] have proposed CloneCloud, which is a system for elastic execution between mobile devices and the cloud, through dynamic application partitioning, where a thread of the application is migrated to a clone of the smartphone in the cloud. Application partitioning is based on static analysis to specify the migration and reintegration points in the application.

Cuervo et.al [6] proposed a system called MAUI, based on code annotations to specify which methods from a class can be offloaded. Annotations are introduced in the source code by the developer during the development phase. At runtime, methods are identified by the MAUI profiler, which performs the offloading of the methods, if the bandwidth of the network and data transfer conditions are ideal. MAUI aims to optimize both the energy consumption and execution time using an optimization solver. However, this annotation method puts an extra burden on the already complex mobile application development phase. Moreover, such annotations can cause unnecessary code offloading that drains energy [5]. To reduce the complexity of the application development

process, we suggest to transfer the entire application processing in the cloud rather than utilizing a design-time code partitioning method. Further, we propose a novel adaptive reward-based learning approach to make smart offloading decisions that can achieve energy efficiency with offloading and also improve application response time.

CHAPTER 3

CHALLENGES WITH OFFLOADING

In spite of existing research highlighting the potential of offloading in mobile devices, current offloading techniques are far from being adopted widely in mobile systems. The implementation of these computation offloading techniques for many real world mobile applications in real-world scenarios has not shown promising results [8], with the mobile device spending more energy in the offloading process than the energy savings achieved due to computing on servers in an offloaded manner.

The granularity of offloading is an important decision. Code partitioning techniques such as in [6] rely on the expertise of the software developer to annotate portions of an application statically to guide the offloading process. Thus the offloading occurs at a sub-application granularity. It is however a difficult task to evaluate runtime properties of code to determine how best to annotate code within an application that can benefit from offloading. Code can often have non-deterministic behavior during runtime (e.g., the run time of a piece of code can be impacted by the thermal and battery/energy conserving strategies employed by the processor or operating system), therefore it is difficult to estimate the suitability of offloading in many cases, especially at a finer, sub-application granularity. Other factors such as the user input, type of application, execution environment, and available memory also create runtime uncertainty, limiting the effectiveness of sub-application partitioning based offloading [5, 8]. We show how we addressed some of these challenges by using a Reinforcement Learning based decision engine to determine when to invoke the offloading process on a mobile device at an application-level granularity.

Offloading decision engines must consider not only the potential energy savings from offloading but also how the response time of the application is impacted by offloading. An effective offloading decision to offload processing to the cloud must reduce energy without significantly increasing response time. Such decisions are heavily impacted by wireless network inconsistency. The power consumed by the network radio interface is known to contribute a considerable fraction of the total device power, and it varies depending on signal strength [32]. With the recent advent of high bandwidth 4G networks, there has been increased interest in the offloading domain, but from our experiments and results presented in later chapters of this thesis, we noticed that 4G consumes more energy than Wi-Fi and 3G. Some of the prior works [13] in this area also confirm this observation.

The network quality of a 4G connection at a mobile device's location greatly affects the battery life. If the device is in the area that does not have 4G coverage, there is no advantage to a 4G interface, and if 4G network search is not disabled, then the radio's search for a non-existent signal will drain the battery quickly. In case of a weak signal, the device uses more power to send and receive data, to and from the network. A strong 4G signal uses less battery, but the biggest problem is the constant switching from 4G to 3G and back again. Also, throughout a typical day, at different times, the performance of a wireless network varies because of changing traffic load on the network. We refer to all such problems due to the mobile network as 'network inconsistency' problems.

To counter the impact of network inconsistency on a mobile device and to optimize the offloading experience, we propose a novel offloading framework based on Reinforcement Learning. This framework not only decides when to offload, but also helps a mobile device select between the different available wireless networks, to achieve consistent improvements by using offloading even in the presence of varying network conditions. In the chapter 6, we describe our framework in detail.

CHAPTER 4

OFFLOADING PERFORMANCE OF MOBILE APPLICATIONS

To gain deeper insights of offloading system it is very important to understand the behavior of real mobile applications. We surveyed various compute intensive applications that are likely to benefit from offloading, as suggested by important publications in this area [3-10]. These applications are power hungry and consume large amount of computational resources. Applications mentioned in relevant publications are as follows: matrix calculations, image processing, web-browsers, torrent downloads, image search, file compressors, online games, language translators, speech recognizers, optical character recognizers, video processing and editing, navigation, face recognition, augmented reality, etc.

In this chapter, we analyzed the performance implications of offloading by comparing two scenarios – one where all computations are performed only on the mobile device without using the cloud at all, and the other where there was a complete reliance on the cloud computation, with minimal computations on the mobile device. We selected five diverse and popular commercially available smartphone applications for our experiment.

Our evaluation primarily focuses on two metrics: (i) battery consumption, and (ii) response time. We have compared the results obtained with these applications for 3G, 4G (HSPA+), and Wi-Fi wireless networks. This comparative study was meant to help us identify various factors that need to be considered for the design of offloading strategies for mobile applications, e.g., identifying the best possible network over which computation can be offloaded to the cloud for any particular

application at a specific location. In the next sections of this chapter we describe the experimental setup and the results in detail. Our findings are discussed in the last section of this chapter.

4.1 Experimental Setup

The power estimation models required to estimate battery consumption were built using power measurements on the LG G3 device running the Android OS version 5.0.1. The contact between the smartphone and the battery was instrumented, and current was measured using the Monsoon Solutions power monitor [18]. The power monitor setup is shown in figure 4.1. The monitor connects to a computer (Lenovo ThinkPad E450, Intel core i5, 4GB memory) running the Monsoon Solutions power tool software, which allows real-time current and power measurements.

Fig. 4.6 Monsoon power monitor setup

We also used the Android Device Bridge (ADB), a software tool to perform battery drain measurements on the Android device. The experiments were performed using AT&T's 3G, 4G (HSPA+) network, and Comcast's 100 Mbps (2.4 GHz Band) Wi-Fi network. ADB was also used

for the response time analysis of smartphone applications. We used Amazon Web Services (AWS) [37] for our mobile-cloud interaction analysis of applications.

We conducted these experiments around Colorado State University's campus in Fort Collins, Colorado, USA. Before conducting our experiments, we followed a few preconditions and rules to ensure meaningful and accurate results while avoiding human error. These rules are as follows:

- Set the device's screen to a consistent and fixed brightness level, to minimize interference from varying screen power consumption (e.g., for different ambient light scenarios) in our measurements; we used the lowest screen brightness level;
- Kill all background processes before measurements;
- Repeat each experiment over 15 iterations to improve result confidence and minimize human error;

We selected five diverse commercially available smartphone applications for our experiments:

- Matrix operations
- Internet browser
- Zipper (file compression app)
- Voice recognition and translation app
- Torrent (file download app)

The next subsection (Section 4.2) gives the details of all the applications considered and the results of their execution for the two scenarios (with and without offloading) outlined earlier. While analyzing these results it is important to note the terminology that we used to define the state of an

application. A “locally run application” is one where the computations are performed on the mobile device only; whereas an “offloaded application” is one that relies entirely on cloud-based computations.

4.2 Experimental Results

4.2.1 Matrix operation app

The matrix calculator app [30] runs on Android based devices. The user is first asked to enter the size of the matrix and all the digits of the matrix manually, and then the user can direct the application to calculate the inverse of that matrix. This application calculates matrix inverse using the adjoint method. For our experiments, we used a set of matrix sizes from 3×3 to 9×9. For the cloud part, we used Amazon Web Services (AWS) cloud computation instance EC2 [37] and web based matrix calculation tools [30]. Figure 4.2 shows the results from our experiment.

The energy consumption in local processing mode is equal to the battery drain in the device while performing the matrix operation; whereas in the cloud mode, energy consumption is the total of battery drain during the idle time of the mobile device while the operation is being performed remotely and the time for data transfer between the mobile device and the cloud. It can be observed that in the local processing mode, the battery consumption of the smartphone increases manifolds with the increasing matrix size, largely because there is an increase in the CPU's energy consumption as the number of floating point operations increase.

Local processing is found to be suitable for operations on small matrices (i.e., 3×3 and 5×5) allowing for low energy consumption on the device and low response time. On the other hand,

offloading the task of matrix calculation to the cloud saves energy and also reduces response time when the matrix size increases.

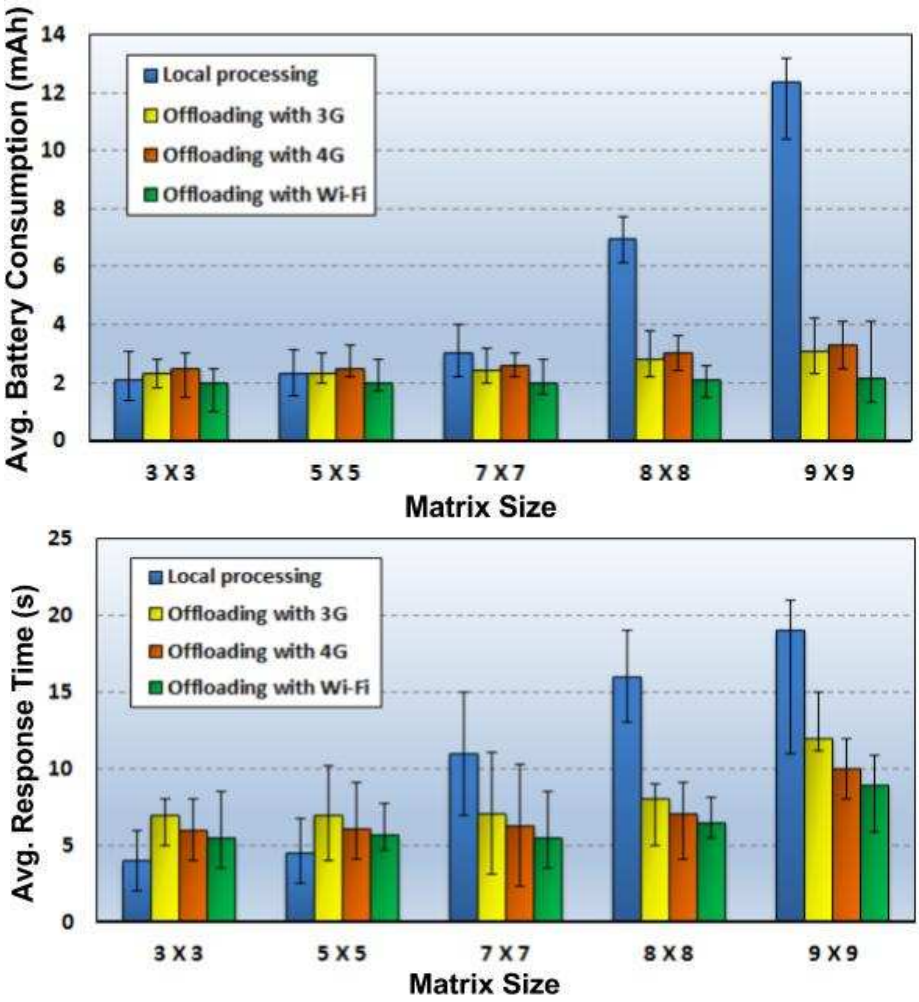

Fig. 4.2 Average battery consumption and average response time on a mobile device for a matrix operation with varying matrix sizes

The device in offloading mode saves maximum energy (and also has minimum response time) when used with Wi-Fi. The results show that 3G performs slightly better than 4G as far as energy is concerned, whereas, 4G gives better response time than 3G for the same operations.

4.2.2 Internet browser app

Cloud-based web browsers use a split architecture where processing of a mobile web browser is offloaded to the cloud partially. This involves cloud support for most browsing functionalities such as execution of JavaScript, image transcoding and compression, and parsing and rendering of web pages. For our experiments, we used the Mozilla Firefox [33] and Puffin [31] browsers. Puffin is a commercially available cloud based mobile browser and Mozilla Firefox is a local browser available from the Google Play store. Our experiments are performed for a data range starting as low as 150 Kib to a session involving 5 MBs of data transfer to load the web pages. Figure 4.3 shows the results obtained by measuring data transfer (response) time and energy consumed by these browsers for loading two different websites: (i) www.yahoo.com and (ii) www.wikipedia.org.

For example, the plots in figure 4.3 show that the response-time/battery-consumption of a browser session with around 3 MB data usage is sometimes more than that of a session which uses 5MB data usage. To counter such network inconsistency problems, we conducted 15 iterations of each experiment across different locations and at different times of the day. In general, our results show that cloud based web browsers are faster but more expensive in terms of energy consumption. For small data transfers it is suitable to use web browsers with local processing to save energy. For a typical user, the data transfer amount during a browsing session does not go beyond 5-6 MBs for a single session. Thus for most websites in typical usage scenarios, the local browser will provide greater energy savings than when using offloading.

We observed that the results obtained fluctuated significantly due to network inconsistency [32]. The response time results indicate that for larger data usage scenarios, offloading can be beneficial. 4G provides lower response time but also consumes more energy than 3G for the offloading

scenarios. Wi-Fi outperforms both 3G and 4G in offloading mode, for response time and energy consumption.

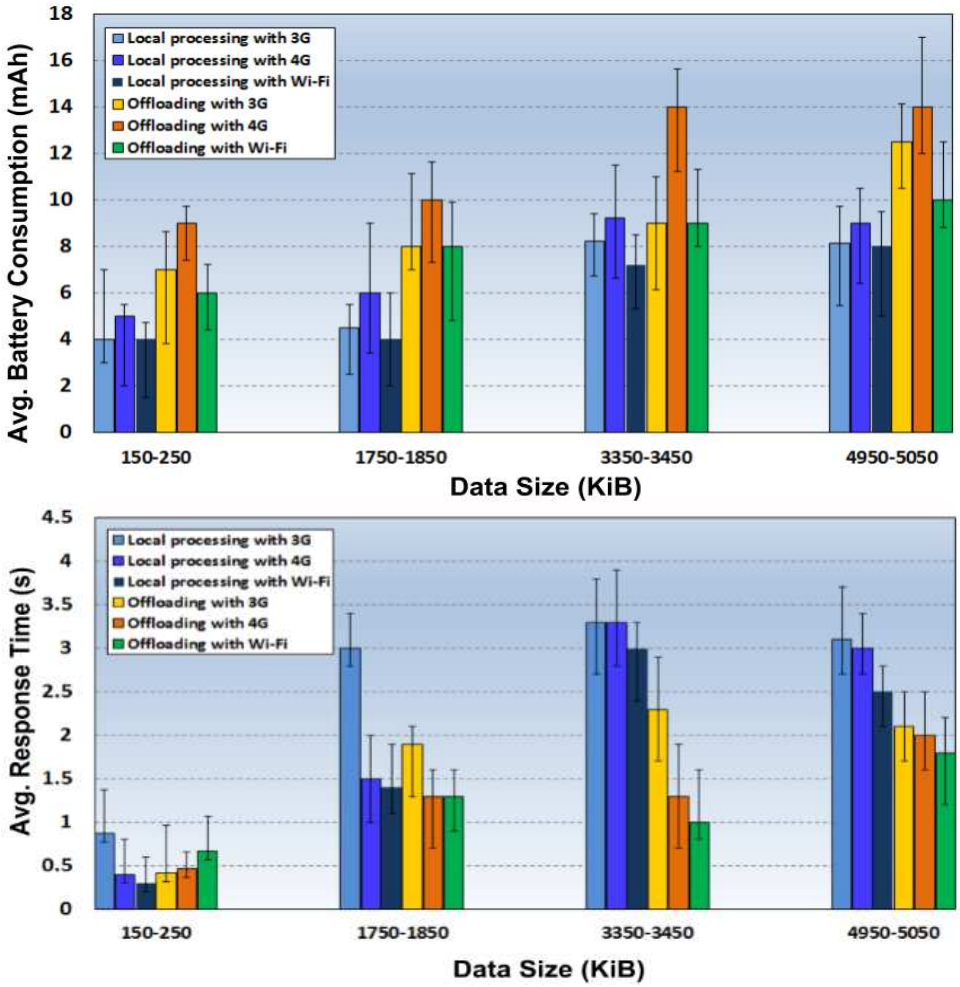

Fig. 4.3 Average battery consumption and response time on a mobile device for an internet browsing session with varying data sizes

4.2.3 Zipper app

Zipping large files in order to compress them is a widely used functionality on most computers. Zipper [22] is an Android app that compresses files locally on a mobile device. For the cloud based file compression, we used an AWS cloud instance and zipping tool available on the web [14].

Figure 4.4 shows the results for energy consumption and the response time when zipping various PDF and Word document files ranging in size from 15MB to 255MB. It can be observed that for the zipping operation, local computation is most efficient in terms of energy consumption. Offloading provides benefits only in response time, and that too only for large file sizes. When offloading, 4G consumes more energy than 3G for smaller file sizes (15-105 MBs) whereas 3G consumes more energy than 4G for larger file sizes (175-255 MBs). 4G is faster than 3G but slower than Wi-Fi. Wi-Fi gives best results in terms of both energy and response time when offloading.

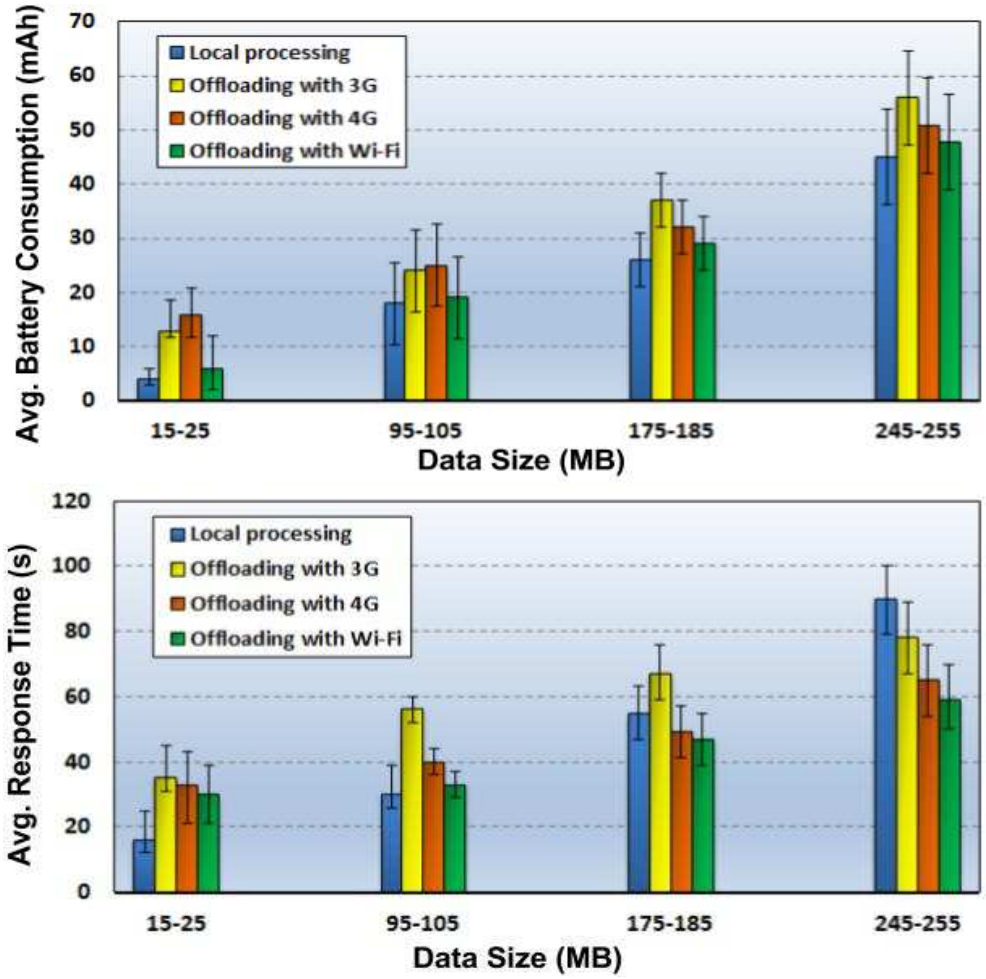

Fig. 4.4 Average battery consumption and response time on a mobile device for zipping/compressing files of varying sizes

4.2.4 Voice recognition and translation app

There are several popular smartphone apps for voice recognition and translation available from app stores, for instance Google Translate [23] for Android and Speak & Translate [24] for iOS. Google Translate is a cloud-based app, which also has an offline translation mode that does local processing on the device with a small neural network. The application allows for downloading an installation package to support the local processing mode. It makes use of the statistical machine translation method, which relies on large amounts of data in order to train a machine translation engine.

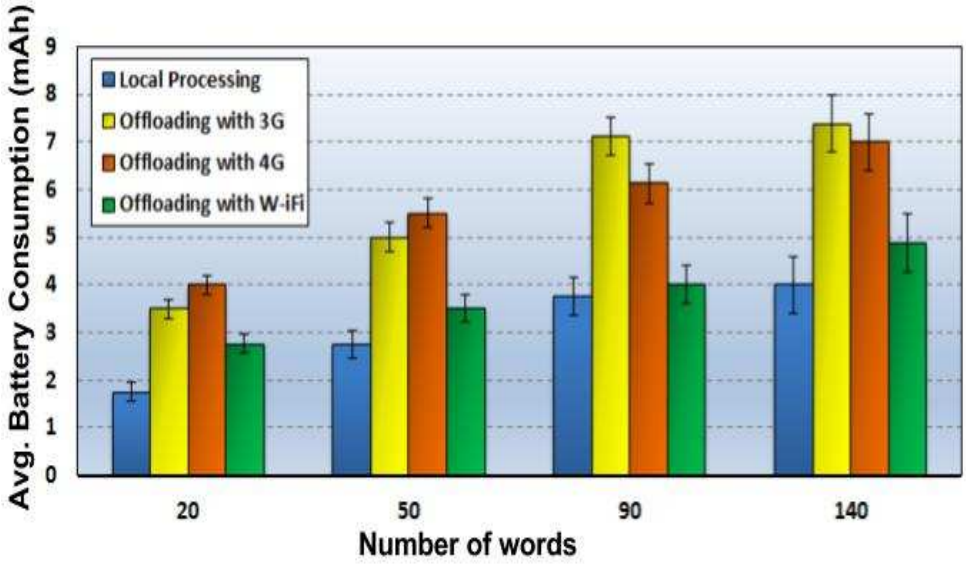

Fig. 4.5 Average battery consumption on a mobile device for voice recognition and translation operations

Figure 4.5 shows the energy consumption of the Google Translate app for a range of words. These measurements were recorded while translating 20-140 words from English (US) to Marathi language. Table 4.1 shows the prediction accuracies for local and offloaded processing. From the results in figure 4.5, we can clearly observe that the local processing mode is more efficient in terms of energy consumption as compared to the cloud processing mode. From table 4.1, it can be seen

that the cloud-processing mode shows better accuracy for voice recognition and translation. This is because the offloaded voice data is processed by more powerful cloud servers, which are capable of running the complex computations of a larger neural network, and other machine learning algorithms for more efficient translation.

Table 4.1 Accuracy of voice recognition and translation app for local vs. cloud processing

Processing	Voice recognition and translation accuracy (%)
Local processing	79.26
Cloud processing	88.51

4.2.4 Torrents app

We used the Android based torrent app known as Flud [34] to perform torrent downloads in local mode. In the cloud mode, a cloud server is used as a BitTorrent client to download torrent pieces on behalf of a mobile device. While the cloud server downloads the torrent, the mobile device switches to the sleep mode until the cloud finishes the torrent processes, and then the cloud uploads because downloading torrent pieces from multiple torrent peers consumes more energy than downloading one burst of pieces from the cloud. the downloaded torrent file in a single process to the mobile device. Kelenyi et al. [16] presented a similar strategy for torrent file download. This strategy saves energy consumption in smartphones

For our experiments, we used torrent file sizes ranging from 25MB to 85MB, with an AWS cloud instance being used for the cloud mode. Figure 4.6 shows the results of our experiments for this application. It is interesting to note that out of all the applications that we consider, offloaded processing proves to be most beneficial (in terms of both energy savings and response time) for the

torrent download application, which is data intensive but not compute intensive. 4G is faster than 3G but slower than Wi-Fi, which is consistent with observations for the other applications. 4G performs slightly better than 3G in terms of energy consumption for higher data sizes (45-85 MBs), but for smaller data sizes 3G is more energy efficient than 4G.

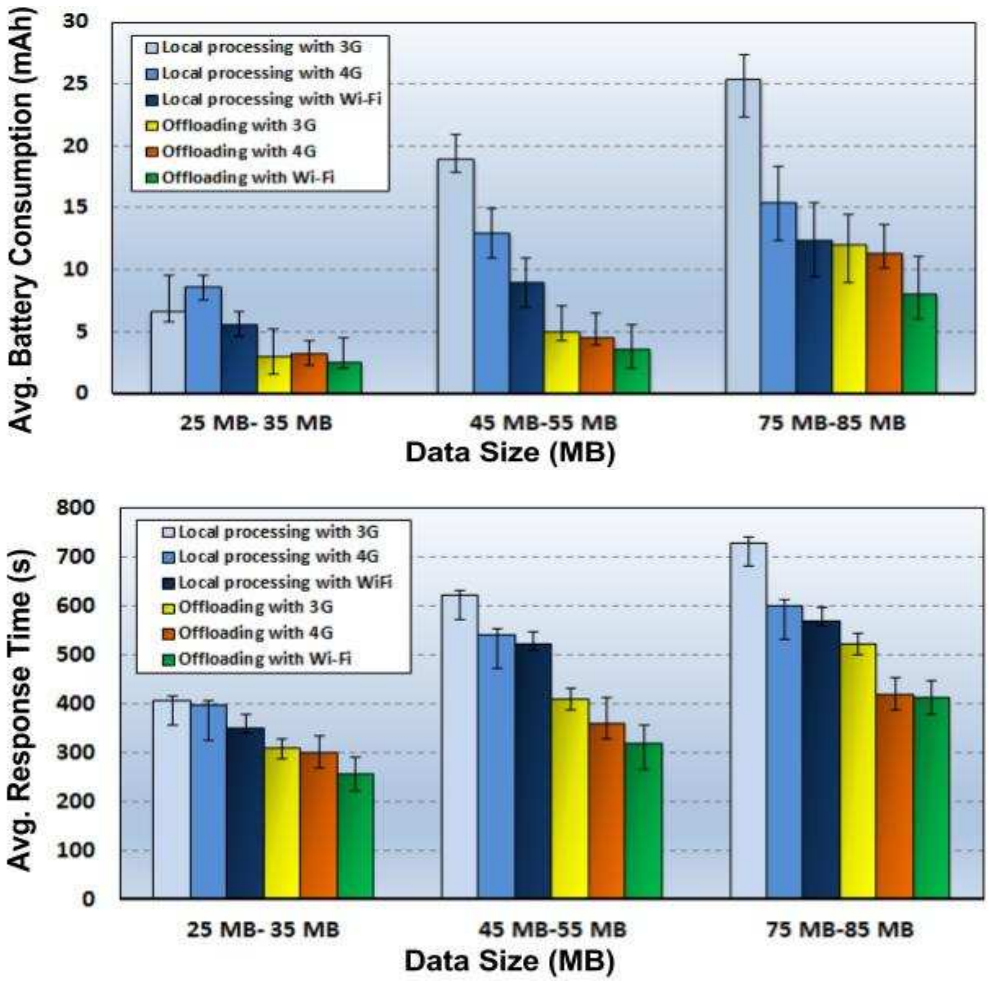

Fig. 4.6 Average battery consumption and response time on a mobile device for torrent file download operations

4.3 Summary of findings

The overall performance when offloading depends on various factors such as the amount of data usage by the application, wireless network signal type and strength, and the functionality of the

application under consideration. In some of the previous publications in this area [4, 19], it was concluded that offloading is useful when an application is compute intensive and at the same time less data intensive. However, we found that this is not always the case. In our experiments with real mobile and cloud based applications, we found that cloud computing can be more beneficial for applications that may not be compute intensive, but are data intensive, e.g., the torrent application. Internet browsers are neither highly compute or data intensive, thus for such applications offloading does not perform well.

To make offloading more practical, it is important to reduce the energy spent in the communication between the mobile device and the cloud. In our experiments, we compared energy consumption in mobile devices connected using different network types (3G, 4G, and Wi-Fi). This comparison shows that choosing the best possible network for offloading is a critical decision. One may assume that because 4G is fastest, we should always rely on it for offloading when Wi-Fi is not available. However, our results indicate that 4G is more power hungry than 3G most of the time.

Network quality is also an important factor that cannot be ignored. We found that a perfect 3G-coverage performs far better as opposed to poor 4G-coverage and vice versa. In the region of cell tower edges or where the coverage of 3G/4G ends, we found that the handover process results in high battery consumption. This is because the phone in such scenarios is constantly searching for the network, frequently scanning the wireless spectrum around it to determine which tower it should tether itself to. The more networks there are available to choose from, the more scans the device must perform. Some apps require a channel to be established between the base station and the mobile device at regular intervals, which also significantly drains the battery.

Another observation is that as 4G generally provides faster data rates than 3G, users tend to consume more data when connected on 4G than 3G; and this change in usage pattern leads to potentially greater battery drain for 4G capable mobile devices. The radio-networking interface in the 4G (or LTE) device is functionally a lot more sophisticated and does a lot more than a 3G interface. This network interface is the single biggest source of battery drain in a mobile device, apart from its display. Unlike the display however, the network interface radio is always on. 4G is particularly energy hungry because most of the 4G devices sold today use multi-input multi-output (MIMO) antenna technology, which supports multiple parallel transmissions (typical 4G phones have two antennas, each of which requires its own power amplifier).

In conclusion, we observed from our experiments on real applications running on a real mobile device that the overall performance of offloading depends on various factors, such as the amount of data usage by the application, network signal type (3G, 4G, Wi-Fi), network signal strength, and the complexity of the functionality of the application under consideration.

CHAPTER 5

ADAPTIVE OFFLOADING

The decision to offload a mobile application to the cloud is a complex one due to the distributed nature and many real-time constraints of the overall system. For making an effective offloading decision, it is important to consider various factors as we discovered after our experimental analysis in the previous chapter. As these factors vary at run-time, there is a need for an adaptive offloading approach that takes the variations of these factors at run-time into consideration when making decisions. A few prior works [10, 11] propose an offloading decision engine that considers the contextual parameters on a device and on the cloud to make an offloading decision adaptively. Figure 5.1 shows an overview of the general framework for utilizing an offloading decision engine mechanism to govern offloading from the device.

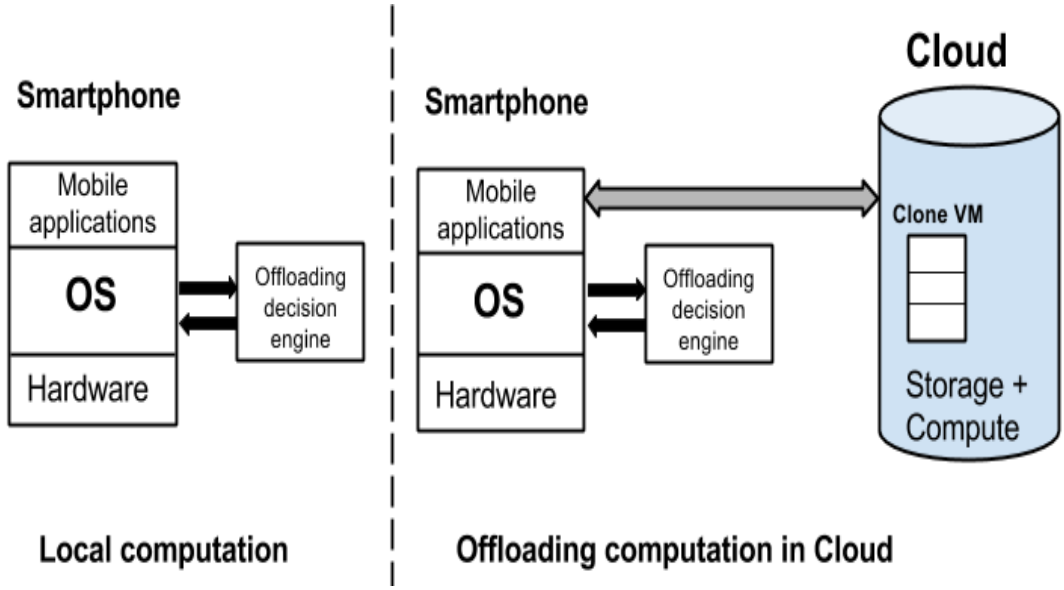

Fig. 5.1 Offloading decision engine mechanism

As an example, Flores et al. [10] proposed a fuzzy decision engine for code offloading. At the mobile platform level, the device uses a decision engine based on fuzzy logic, which is utilized to combine *n* number of variables (e.g., application data size, network bandwidth), which are to be obtained from the overall mobile cloud architecture. Fuzzy logic decision engine works in three steps namely: fuzzyfication, inference, and defuzzification as shown in figure 5.2.

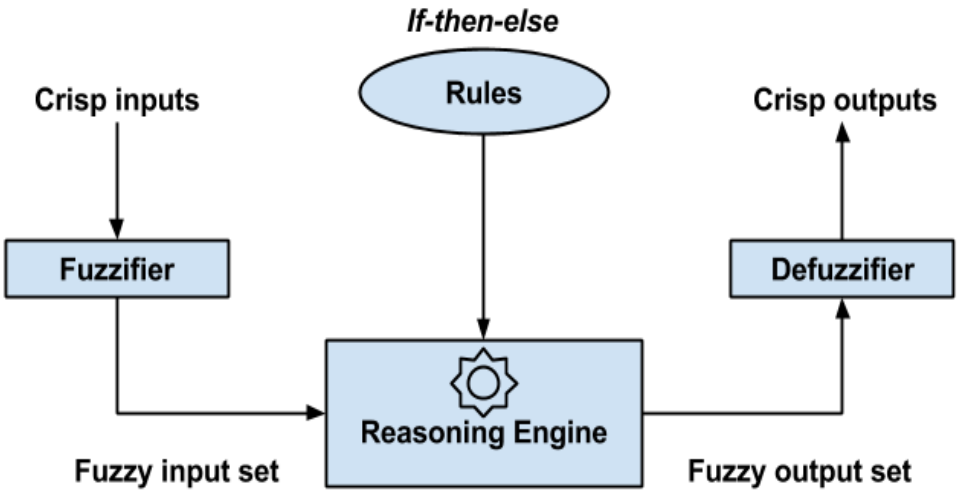

Fig. 5.2 Fuzzy decision engine mechanism [10]

Fig. 6.1 Reinforcement Learning (RL) based middleware framework for

In fuzzification, input data is converted into linguistic variables, which are assigned to a specific membership function. A reasoning engine is applied to the variables, which makes an inference based on a set of rules. Lastly, the outputs from the reasoning engine are mapped to linguistic variable sets again in the defuzzification step. This offloading decision engine in [10] assumes a consistent network performance during offloading. However, as observed in our experiments, such consistency is difficult to achieve because of frequent mobile user movements and variable network quality (due to factors such as location of the device and load on the network [32]). Moreover, the offloading decision engine in [10] mainly emphasizes energy savings; however response time is

also a crucial metric for various applications that should not be ignored, otherwise user quality of service degradations can become so severe that any effort to save energy becomes irrelevant.

In the next chapter, we describe our reward-based middleware framework for adaptive offloading that overcomes the challenges mentioned above, to make more efficient decisions related to when and how to offload applications from a mobile device to the cloud.

CHAPTER 6

MIDDLEWARE FRAMEWORK FOR EFFICENT OFFLOADING OF MOBILE APPLICATIONS

To simplify the mobile application development process and at the same time avoid problems caused by hard coded annotations, our framework proposes to transfer all the computation to the cloud instead of partial offloading. Our framework involves a novel decision engine on the mobile device that works together with a clone virtual machine (VM) of the mobile software environment to execute applications on cloud servers. Figure 6.1 shows a high level overview of the proposed framework. The framework is implemented at the middleware level in the software stack of the Android OS, and runs in the background as an Android service. As a result, our framework requires no changes to any of the applications or the Android OS. The runtime monitor component

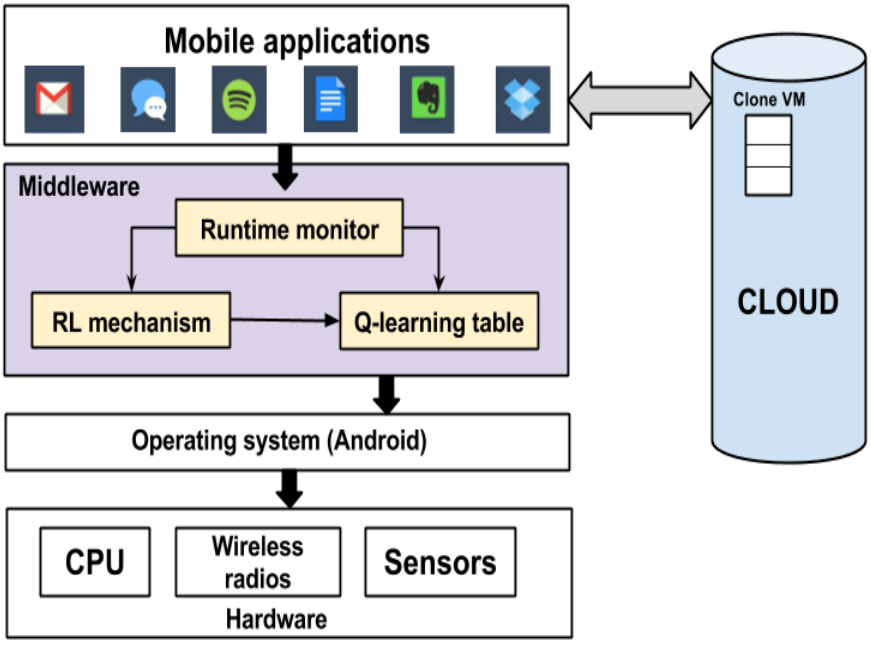

Fig. 6.1 Reinforcement Learning (RL) based middleware framework for efficient application offloading to cloud

periodically triggers the Reinforcement Learning (RL) module to generate/update a Q-learning table. At any time, this Q-table contains information to guide the decision for when and how to offload an application to the cloud, depending on multiple factors. The remainder of this section provides a detailed overview of the RL mechanism and our algorithm to generate and use the Q-table.

Reinforcement Learning (RL) is an unsupervised learning approach, which focuses on learning by interacting with an environment. In supervised learning a training set of correctly identified observations is available which is used to train a prediction model. RL differs from supervised learning in that correct input/output pairs of identified observations are never presented. In RL the state-action value function is a function of both state and action, and its value is a prediction of the expected sum of future reinforcements. The state-action value function is referred to as the Q function [36].

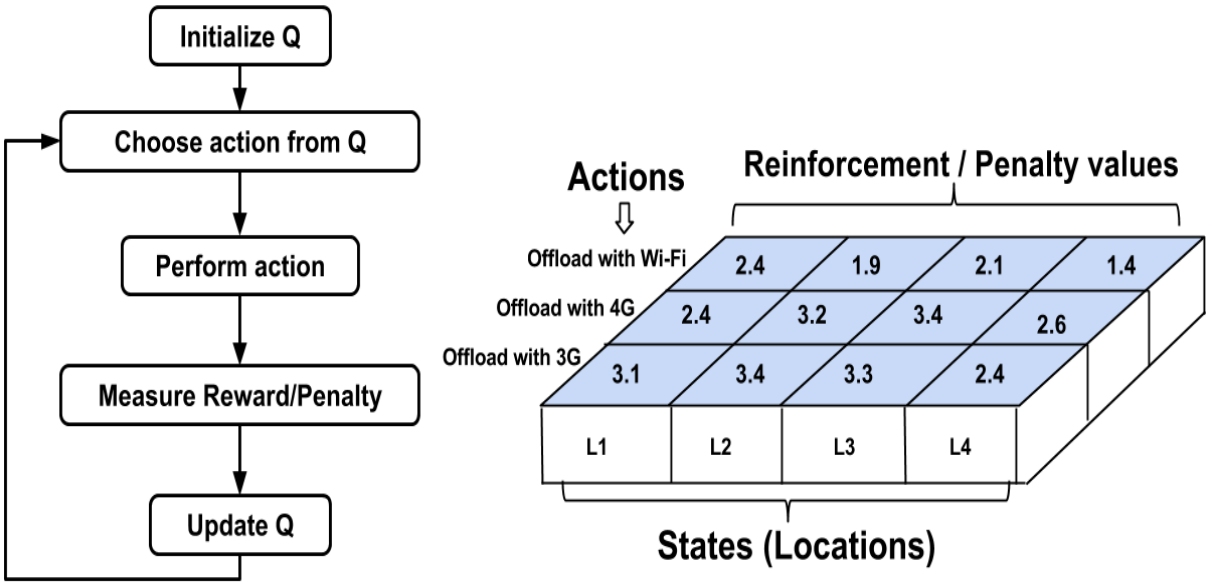

Fig. 6.2 Q-learning flow and Q-table

Figure 6.2 summarizes how a typical Q-learning reinforcement algorithm works. Q-learning is a reward based mechanism that generates a Q table with reinforcement or penalty values as shown in the figure 6.2. The figure illustrates a vector of Q table where the possible actions are offloading with 3G, 4G or Wi-Fi network, when the user is at different locations L1-L4. Actions are chosen and the penalty values are calculated for respective actions to update the Q table.

When the system is at a defined state s_t at time t. Upon taking action a_t from that state, we observe the one step reinforcement r_{t+1}, and the next state becomes s_{t+1}. This continues until we reach a goal state, K steps later. The return R_t from state s_t is shown in equation (1).

$$R_t = \sum_{k=0}^{K} r_{t+k+1} \qquad (1)$$

The objective is to find the actions a_t that maximize (or minimize) the sum of reinforcements or rewards r_t. This can be reduced to the objective of acquiring the Q function $\mathrm{Q}(s_t, a_t)$, which predicts the expected sum of future reinforcements; where the correct Q function determines the optimal next action. So, the RL objective [36] is to make this approximation as accurate as possible:

$$\mathrm{Q}(s_t, a_t) \approx \sum_{k=0}^{\infty} r_{t+k+1} \qquad (2)$$

The Q function stores reinforcement values for each state and action pair of the system. Equation (2) formulates the RL for multi-step decision problem, for example a predicting sequential positions of Tic-Tac-Toe game [36]. In our middleware framework we use RL for a single-step decision problem as there are no sequential states which are dependent on previous state of the system, formulated in equation (3):

$$\mathrm{Q}(s_t, a_t) \approx \sum_{t=1}^{n} r_t \qquad (3)$$

The Q function is referred by system to select the optimal action a_t, in state s_t:

$$a_t = argmin\ Q(s_t, a) \qquad (4)$$

6.1 RL algorithm to generate Q function

The state of a mobile system is defined using the contextual information of the device such as location, network type and, network strength. These contextual factors were chosen because we consider them to be crucial for better offloading experience. The runtime monitor extracts the contextual information of the device to form state values of the system. For example, a mobile device is at location L1, it has access to 3G network type with 'strong' network strength. From this state, if an application processing needs to be offloaded then the Q function is called to select the right network that would result in the least penalty in terms of energy or response time or both. Penalty values are directly proportional to the battery units consumed and the response time of a task. Less penalty means less energy consumed and less response time for application processing, resulting in enhanced user experience.

In our framework, the following state and action values are used to generate the Q function:

Set of state values (discrete values):

- Location = L1, L2, L3,……, Ln
- Network carrier = 3G, 4G, Wi-Fi
- Network strength = Strong, Medium, Weak
- Data Size = data_small, data_medium, data_large

Set of action values

- Offload using 3G network
- Offload using 4G network
- Offload using Wi-Fi network

The location L1-Ln can be any geographic area where the user uses offloading application, for example office, home, etc. These state action pairs are user defined and more pairs can be added to account for factors which might affect offloading, for example we can add 'Time period' as another state value, as it is observed that network performance is slow at certain time of day when the user load is high. Bigger set of state value pairs means larger Q function resulting in increased overhead to manage it. Following algorithm shows steps to generate Q function:

Algorithm: to generate Q function:

Inputs: user location, network access type, network strength

1. Access user's contextual information such as, location, network_carrier, network_strength.
   while mobile network is available and user is at location L1
   do:
2. Activate 3G radio interface of the device
3. Upload a file (data_size = data_small) in the cloud. For this operation, measure battery units consumed P_{b3G} and response time P_{t3G}
4. Calculate the penalty P with the help of equations:
   $$P_{3G} = P_{b3G} * x + P_{t3G} * y$$
5. Form a Key-Value pair as follows:
   {location-data_size-network_carrier-network_strength: Penalty}
   where, (Key = location-data_size-network_carrier-network_strength,
   Value = Calculated Penalty value P in step 4)
6. Update the Q-table with the calculated penalty values
7. Repeat steps 3-6 above for: data_size = data_medium and data_large
8. Repeat steps 3-7 above for 4G and Wi-Fi connection if available
   end while
9. Go to step 1 when user location changes to L2

Output: Q table

At the beginning (step 1), when the mobile device is at user location L1, the runtime monitor accesses contextual information from the device such as location, network carrier type and, network strength. A network operation of uploading a data file from a mobile device to cloud is performed, using available network carrier (step 3). The battery units consumed and total response time taken for this operation are measured. In the multiple iterations of the uploading operation we used varying data sizes with all available network radio interface antennas (3G, 4G, Wi-Fi) activated one by one. For each of these uploading operations the runtime monitor measures the battery units consumed and response time to complete the operation. The Q table is updated (step 6) with the penalty values calculated using equations (5), (6) and, (7). In our RL framework the reinforcement values are penalty values P_{3G}, P_{4G} and P_{WiFi}. The Q table is formed using these key-value pairs where the state of the system is used to form keys and calculated penalties are used as values as shown in figure 11. The series of possible individual penalty values are shown in Table 2.

Table 6.1 Penalty values in RL algorithm

Penalty values	Offload using 3G	Offload using 4G	Offload using Wi-Fi
Battery (P_b)	P_{b3G}	P_{b4G}	P_{bWiFI}
Response time (P_t)	P_{t3G}	P_{t4G}	P_{tWiFI}
Total penalty	P_{3G}	P_{4G}	P_{WiFI}

$$P_{3G} = P_{b3G} * x + P_{t3G} * y \qquad (5)$$

$$P_{4G} = P_{b4G} * x + P_{t4G} * y \qquad (6)$$

$$P_{WiFI} = P_{WiFI} * x + P_{WiFI} * y \qquad (7)$$

In equations (5), (6) and, (7), to optimize battery consumption and response time we used weights x and y respectively with penalty values. Variables x and y are multiplying factors with values between 0 and 1, used to optimize either battery consumption or response time or both. For optimized battery consumption we found following weight values suitable: $x = 0.3,\ y = 0.7$, whereas, for optimized response time we used: $x = 0.7, y = 0.3$. For our experiments discussed in chapter 7 we used $x = 0.5, y = 0.5$ for both optimized battery and response time.

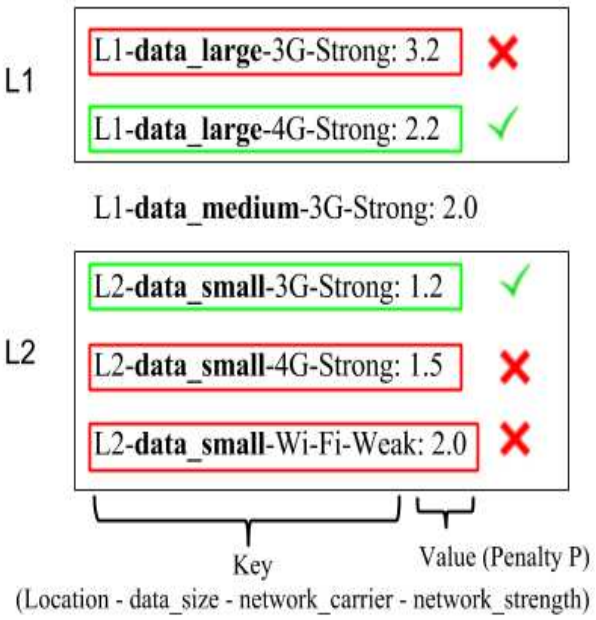

Fig. 6.3 Decision making using Q-table (vector of key value pairs)

Figure 6.3 briefly shows the decision making process with the help of two simple scenarios. For a data intensive application at location L1 we have 3G and 4G networks available as shown in first two lines of Q table in figure 6.3. The penalty value for 4G at location L1 is lesser, therefore 4G network is selected for offloading the application to cloud. For a less data intensive application at location L2, out of all the networks available 3G is selected because Wi-Fi has weak signal strength with higher penalty and 4G also has a higher penalty.

CHAPTER 7

EXPERIMENTAL RESULTS

To evaluate the efficacy of our proposed framework we conducted a series of experiments with three applications that we discussed in chapter 4. We implemented our middleware framework and its decision engine on an Android-based mobile device. To form the Q function of our RL algorithm, real user data was collected at 3 different geographical locations around the Colorado State University campus area, in Fort Collins, Colorado. In this chapter we discuss our results in detail and compare our work with the fuzzy logic decision engine proposed by Flores et al. [10] which we discussed in chapter 5. We used data sizes with 25 MBs, 250 MBs and, 500 MBs files which represent variables data_small, data_medium and, data_large respectively in the Q table algorithm discussed in chapter 6.

Figure 7.1 shows the results for the Matrix operations app with RL based engine and fuzzy logic based decision engine. Similarly figure 7.2 shows the results for the Zipper app and figure 7.3 shows results for the Torrent app. In all the scenarios, the task of a decision engine is to choose to tune to the network which gives best possible results. In these figures, the red trendline shows the average battery consumption and average response time for fuzzy decision engine [10] whereas the green trendline shows results of our RL-based middleware framework.

We have also shown the offloading results for varying networks obtained in chapter 4 as a reference in all these figures. Our results show that for less data intensive operations the results of RL and fuzzy logic overlap. In the case of zipper application for lower data sizes fuzzy logic shows better results, this might be because the Q-table generated using our RL algorithm uses 25 MBs as the minimum data size. For higher data sizes and computations the RL method gives improved

battery consumption and response time. For any data size lower than that there is insufficient predictable results. This can be improved using a wide range of data files in RL algorithm.

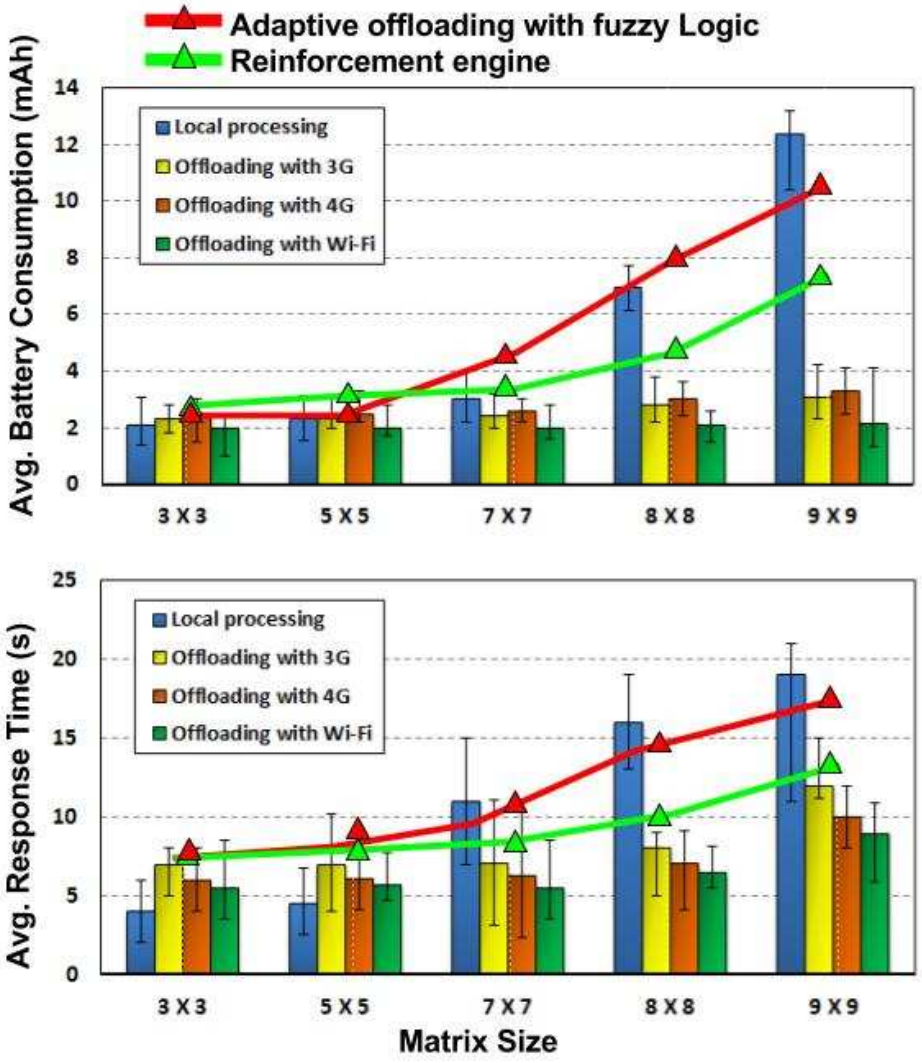

Fig. 7.1 Average battery consumption and response time of Matrix operations app with learning methods

Figure 7.4 shows prediction accuracy of both learning methods. It can be observed that our RL-based engine has better prediction accuracy which is crucial for making effective offloading decisions. The overall performance of offloading depends on various factors, such as the amount of

data usage by the application, network signal type (3G, 4G and, Wi-Fi) and network signal strength, and the complexity of the functionality of the application under observation. By considering all of these individual factors in the decision process, unlike the fuzzy logic approach from [10], and by utilizing a more sophisticated and powerful learning algorithm, our framework is able to achieve notably better results compared to [10]. Our results show that proposed RL based offloading system can save up to 30% battery power with up to 25% better response time as compared to fuzzy logic based system.

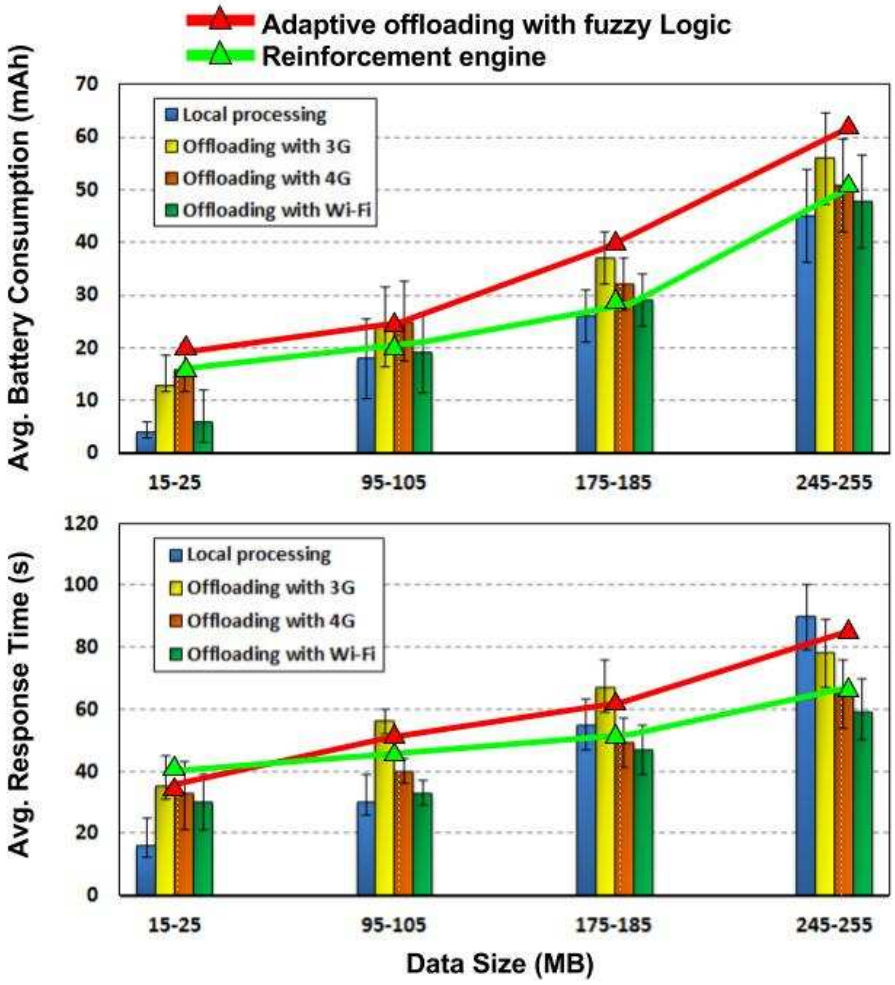

Fig. 7.2 Average battery consumption and response time of Zipper app with learning methods

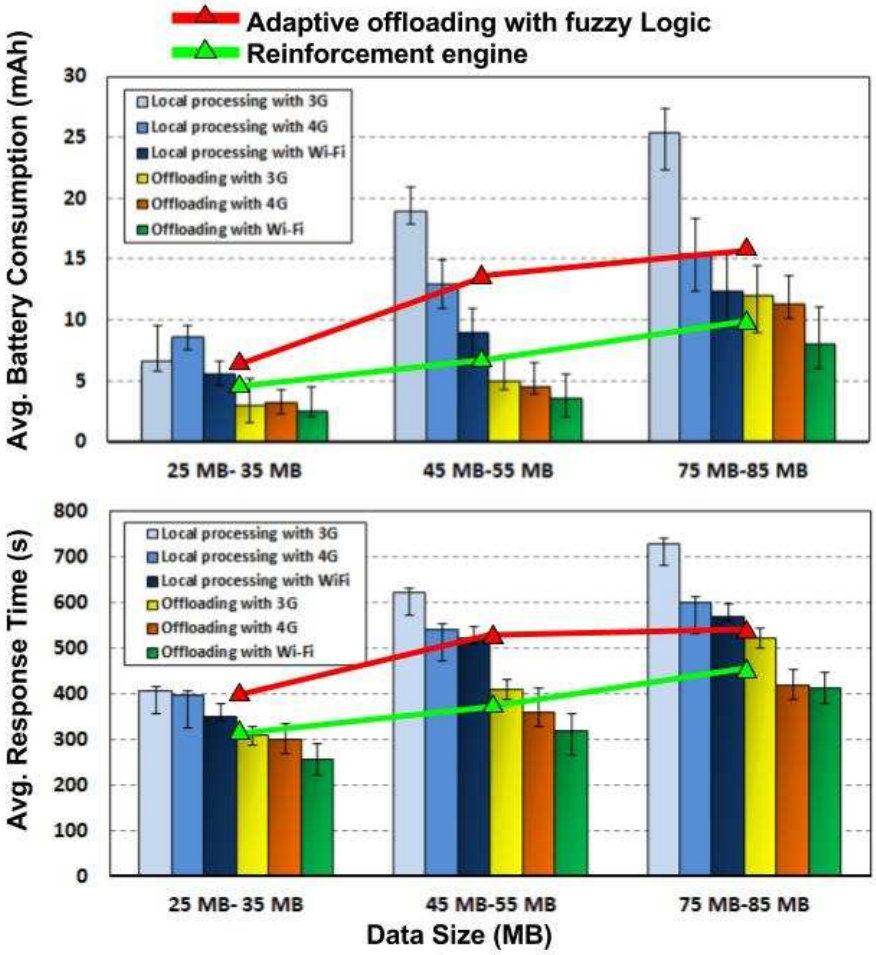

Fig. 7.3 Average battery consumption and response time of Torrent app with learning methods

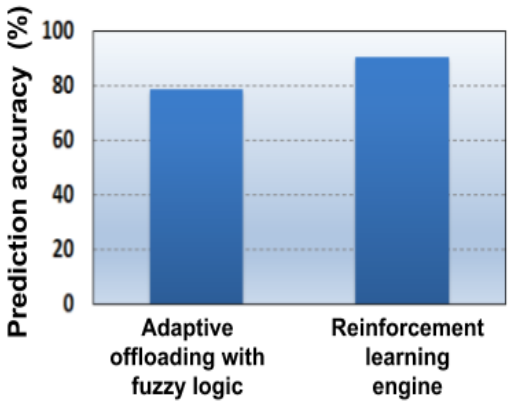

Fig. 7.4 Prediction accuracy of learning methods

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this thesis we analyzed real mobile applications in detail to study benefits of application offloading. We found that overall performance with offloading depends on factors such as the amount of data and type of usage, available network carrier and signal strength, etc. These factors should be considered while making a decision to offload a mobile application. To make offloading more practical, it is important to reduce the energy spent in the communication between the mobile device and the cloud. In our experiments, we compared energy consumption in mobile devices for varying network types (3G, 4G and Wi-Fi). This comparison shows that choosing the best possible network for offloading is crucial. In this paper, we presented an intelligent mobile network aware middleware framework based on Reinforcement Learning for energy efficient offloading in smartphones. Our results show that we can save up to 20%-30% battery power by using the proposed offloading system.

8.2 Future Work

Offloading is far from being adopted in the design of current mobile architectures due to many challenges in this field. Strategies described in this thesis show promising energy savings, however, much work can be done to improve the offloading strategies.

In this thesis we have used only one mobile network carrier (AT&T) to obtain the offloading results, it will be interesting to see the comparison between multiple network carriers. Even in 4G

network, there are different types of available technologies such as HSPA+ and LTE, in this thesis we have used AT&T's HSPA+. In addition, there are different technologies in implementing 4G LTE for different network carriers. This study can be further extended to test the results with all variations of 4G network.

Finally, we would like to implement and test more software applications, such as an image search to the cloud, video processing etc. to gain insights into versatile offloading scenarios. Although there are always improvements to be made in the field of software and energy optimization for mobile embedded systems, the work presented in this thesis brings us one step closer to being able to improve the performance and battery lifetime of smartphone while computation offloading.

REFERENCES

[1] 'The Statistical Portal', 2016. [Online]. Available: www.statista.com/statistics, [Accessed: 7 Dec 2016].

[2] F. A. Ali, P. Simoens, T. Verbelen, P. Demeester, B. Dhoedt, "Mobile device power models for energy efficient dynamic offloading at runtime." Journal of Systems and Software 113, 2016.

[3] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, & A. Schmidt, "Micro-blog: sharing and querying content through mobile phones and social participation." Proc. ACM Mobisys, 2008.

[4] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" Computer, vol. 43, 2010.

[5] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: from concept to practice and beyond", Communications Magazine, IEEE, vol. 53, 2015.

[6] 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 Proc. ACM Mobisys, 2010.

[7] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in Proc. ACM EuroSys, 2011.

[8] H. Flores and S. Srirama, "Mobile code offloading: should it be a local decision or global inference?" Proc. ACM Mobisys, 2013.

[9] 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," Proc. IEEE INFOCOM, 2012.

[10] H. R. Flores and S. Srirama, "Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning," Proc. ACM Mobisys, 2013.

[11] A. Khairy, H. H. Ammar, and R. Bahgat, "Smartphone energizer: Extending smartphone's battery life with smart offloading," in IWCMC, 2013 9thInternational, IEEE, 2013.

[12] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in Mobile Computing, Applications, and Services, Springer, 2012.

[13] A. Sivakumar, V. Gopalakrishnan, S. Lee, S. Rao, S. Sen, & O. Spatscheck, "Cloud is not a silver bullet: A case study of cloud-based mobile browsing," Proc. ACM Mobisys, 2014.

[14] 'Ezyzip: The simple online zip tool', 2016. [Online]. Available: http://www.ezyzip.com/ [Accessed: 5 Dec 2016].

[15] 'Online-convert.com', 2016. [Online]. Available: http://archiv-e.online-convert.com/convert-to-zip [Accessed: 8 Aug 2016].

[16] I. Kelenyi and J. K. Nurminen, “Cloudtorrent-energy-efficient bittorrent content sharing for mobile devices via cloud services,” Proc. IEEE CCNC, 2010.

[17] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, “A close examination of performance and power characteristics of 4g lte networks,” Proc. ACM Mobisys, 2012.

[18] ‘Monsoon Solutions Inc., official website’, 2016. [Online]. Available:http://www.msoon.com/LabEquipment/Power-Monitor, [Accessed: 9 Nov 2016].

[19] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak, “Energy cost models of smartphones for task offloading to the cloud,” IEEE Trans. Emerg. Topics Comput. vol. 3, Sep. 2015.

[20] B. K. Donohoo, C. Ohlsen, S. Pasricha, Y. Xiang, & C. Anderson, “Context-aware energy enhancements for smart mobile devices.” IEEE Transactions on Mobile Computing, 2014.

[21] B. K. Donohoo, “Thesis on Machine learning techniques for energy optimization in mobile embedded systems.” 2012.

[22] ‘Zipper Android App’, 2016. [Online]. Available: https://play.google.com/store/apps/details?id=org.joa.zipperplus&hl=en, [Accessed: 8 Dec 2015].

[23] ‘Google Translate Android App’, 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.translate&hl=en, [Accessed: 3 Dec 2016].

[24] ‘Speak & Translate iOS app’, 2016. [Online]. Available: https://itunes.apple.com/us/app/speak-translate-free-live/id804641004?mt=8, [Accessed: 2 Nov 2016].

[25] M. E. Khoda, M. A. Razzaque, A. Almogren, M. M. Hassan, A. Alamri, & A. Alelaiwi, “Efficient Computation Offloading Decision in Mobile Cloud Computing over 5G Network.” Mobile Networks and Applications 2016.

[26] N. Vallina-Rodriguez, and J. Crowcroft. “Energy management techniques in modern mobile handsets.” Communications Surveys & Tutorials, IEEE 15.1, 2013.

[27] J. Flinn, S. Park, & M. Satyanarayanan. “Balancing performance, energy, and quality in pervasive computing.” Proc. IEEE ICDCS 2002.

[28] T. Verbelen, P. Simoens, F. De Turck, B. Dhoedt, “AIOLOS: Middleware for improving mobile application performance through cyber foraging.” Journal of Systems and Software 85.11 (2012).

[29] ‘Fuzzy Logic vs Machine Learning an Investigation’, 2016. [Online]. Available: http://www.kev.pulo.com.au/ai/fuzzy-ml_report, [Accessed: 22 Nov 2016].

[30] ‘Matrix Calculator app at Google play store’, 2016. [Online]. Available:https://play.google.com/store/apps/details?id=ru.alex-anderskokov.matrix&hl=en, [Accessed: 3 Nov 2016].

[31] ‘Puffin Web Browser’, 2016, [Online]. Available: http://www.puffinbrowser.com/, [Accessed: 3 Nov 2016].

[32] J. Li, K. Bu, X. Liu, & B. Xiao, “Enda: Embracing network inconsistency for dynamic application offloading in mobile cloud computing.” Proc. ACM SIGCOMM, 2013.

[33] ‘Mozilla Firefox’, 2016, [Online]. Available: https://www.moz-illa.org/en-US/firefox/android [Accessed: 6 Dec 2016].

[34] ‘Fuld – Torrent Downloader app’, 2016, [Online]. Available: https://play.google.com/store/apps/details?id=com.delphicoder.flud&hl=en, [Accessed: 3 Nov 2016].

[35] ‘Machine Learning website’, 2016, [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning, [Accessed: 3 Aug 2016].

[36] ‘C. Anderson, Introduction to Machine Learning. Website’, 2016, [Online]. Available: http://www.cs.colostate.edu-/~anderson/cs545/index.html/doku.php, [Accessed: 7 Aug 2016].

[37] ‘Amazon web services (AWS)’, 2016, [Online]. Available: https://aws.amazon.com/, [Accessed: 3 Nov 2016].

[38] ‘Dreamstime – stock images’, 2017, [Online]. Available: https://www.dreamstime.com [Accessed: 1 Mar 2017]

APENDIX A

SOURCE CODE

This section presents the majority of the source code for the implementation of the two strategies namely reinforcement learning and fuzzy logic decision engine. Section A.1 provides the source code file for the offloading decision engine in Android, section A.2 provide the source code file for fuzzy logic and sections A.3 provide the source code files for the reinforcement learning python code.

A. RL1_Mainoffloadingappactivity.java

```
package com.example.aditya.smartoffloadingapp;

import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.Spinner;
import android.widget.TextView;

import static com.example.aditya.smartoffloadingapp.R.id.MLAlgorithm;


public class MainOffloadingAppActivity extends ActionBarActivity {
    public        final        static        String        EXTRA_MESSAGE        =
"com.example.aditya.smartoffloadingapp.MESSAGE";
    public        final        static        String        EXTRA_MESSAGE1        =
"com.example.aditya.smartoffloadingapp.MESSAGE1";


/*    private Spinner spinner, spinnerApp, spinnerCPU;
    private static final String[]paths = {"Fuzzy Logic", "RL", "RL with NN",
"Classification"};
    private static final String[]pathsApp = {"Fuzzy Logic", "RL", "RL with NN",
"Classification"};
    private static final String[]pathsCPU = {"Fuzzy Logic", "RL", "RL with NN",
"Classification"};
**/
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main_offloading_app);


    /*        Spinner spinner, spinnerApp, spinnerCPU;
            String[]paths = {"Fuzzy Logic", "RL", "RL with NN", "Classification"};
            String[]pathsApp = {"Fuzzy Logic", "RL", "RL with NN", "Classification"};
            String[]pathsCPU = {"Fuzzy Logic", "RL", "RL with NN", "Classification"};
    **/

    /*        spinner = (Spinner)findViewById(R.id.spinner);
            spinnerApp = (Spinner)findViewById(R.id.spinnerApp);
            spinnerCPU = (Spinner)findViewById(R.id.spinnerCPU);

            ArrayAdapter<String>adapter                         =                         new
ArrayAdapter<String>(MainOffloadingAppActivity.this,
                    android.R.layout.simple_spinner_item,paths);
            ArrayAdapter<String>adapterApp                       =                         new
ArrayAdapter<String>(MainOffloadingAppActivity.this,
                    android.R.layout.simple_spinner_item,pathsApp);
            ArrayAdapter<String>adapterCPU                       =                         new
ArrayAdapter<String>(MainOffloadingAppActivity.this,
                    android.R.layout.simple_spinner_item,pathsCPU);


adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

adapterApp.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

adapterCPU.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

            spinner.setAdapter(adapter);
            spinnerApp.setAdapter(adapterApp);
            spinnerCPU.setAdapter(adapterCPU);


    **/
    /*        spinner.setOnItemSelectedListener(this); **/

        }


        public void onButtonClick(View view) {

            Spinner   spinner   =   (Spinner)findViewById(R.id.spinner);   //offloading
mechanism
            String offloadingMechanismType = spinner.getSelectedItem().toString();

            CheckBox              responseCheckbox              =              (CheckBox)
findViewById(R.id.CheckBoxResponse);//checkbox
            boolean bRequiresResponse = responseCheckbox.isChecked();

            Spinner   spinnerApp   =   (Spinner)findViewById(R.id.spinnerApp);//Select
Application
            String appType = spinnerApp.getSelectedItem().toString();
            Spinner   spinnerLocation   =   (Spinner)findViewById(R.id.spinnerLocation);
//Matrix operation
            String LocationType = spinnerLocation.getSelectedItem().toString();

            Spinner   spinnerCPU   =   (Spinner)findViewById(R.id.spinnerCPU);   //Matrix
operation
            String CPUinstanceType = spinnerCPU.getSelectedItem().toString();
```

```
   /*                                          ArrayAdapter<String>adapter     =     new
ArrayAdapter<String>(MainOffloadingAppActivity.this,
                    android.R.layout.simple_spinner_item,paths);
            ArrayAdapter<String>adapterApp                      =                   new
ArrayAdapter<String>(MainOffloadingAppActivity.this,
                    android.R.layout.simple_spinner_item,pathsApp);
            ArrayAdapter<String>adapterCPU                      =                   new
ArrayAdapter<String>(MainOffloadingAppActivity.this,
                    android.R.layout.simple_spinner_item,pathsCPU);


adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

adapterApp.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

adapterCPU.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

            spinner.setAdapter(adapter);
            spinnerApp.setAdapter(adapterApp);
            spinnerCPU.setAdapter(adapterCPU);

   **/


   if(offloadingMechanismType.equals("Fuzzy Logic")) {

       Intent fuzzyscreen = new Intent(this, FuzzyLogicDisplay.class);

   /*    EditText editText = (EditText) findViewById(R.id.dataEdit); **/

       Spinner spinnerMechanismText = (Spinner)findViewById(R.id.spinner);
       Spinner spinnerAppText = (Spinner)findViewById(R.id.spinnerApp);


       String messageMechanism = spinnerMechanismText.getSelectedItem().toString();
       fuzzyscreen.putExtra(EXTRA_MESSAGE, messageMechanism);

       String messageApp = spinnerAppText.getSelectedItem().toString();
       fuzzyscreen.putExtra(EXTRA_MESSAGE1,messageApp);

       startActivity(fuzzyscreen);

   }


       }

       @Override
       public boolean onCreateOptionsMenu(Menu menu) {
           // Inflate the menu; this adds items to the action bar if it is present.
           getMenuInflater().inflate(R.menu.menu_main_offloading_app, menu);
           return true;
       }

       @Override
       public boolean onOptionsItemSelected(MenuItem item) {
           // Handle action bar item clicks here. The action bar will
           // automatically handle clicks on the Home/Up button, so long
           // as you specify a parent activity in AndroidManifest.xml.
           int id = item.getItemId();

           //noinspection SimplifiableIfStatement
           if (id == R.id.action_settings) {
```

```
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

B. Fuzzylogicdisplay.java

```
package com.example.aditya.smartoffloadingapp;

import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;


public class FuzzyLogicDisplay extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_fuzzy_logic_display);

        Intent fuzzyintent = getIntent();
        String                                message                                =
fuzzyintent.getStringExtra(MainOffloadingAppActivity.EXTRA_MESSAGE);
        String                                message1                               =
fuzzyintent.getStringExtra(MainOffloadingAppActivity.EXTRA_MESSAGE1);

        TextView t1 = (TextView) findViewById(R.id.FuzzyAlgorithmDisplay);
        t1.setText(message);

        TextView t2 = (TextView) findViewById(R.id.FuzzyAppDisplay);
        t2.setText(message1);




/* create TextView Object **/
/*        TextView textView = new TextView(this); */
/* Set the text size and message */
/*        textView.setTextSize(40); */
/*        textView.setText(message); */
/*add the TextView as the root view of the activity's layout by passing it to
setContentView()**/
/*        setContentView(textView); */
/*        setContentView(R.layout.activity_fuzzy_logic_display); **/
    }



/*
```

```
        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            // Inflate the menu; this adds items to the action bar if it is present.
            getMenuInflater().inflate(R.menu.menu_fuzzy_logic_display, menu);
            return true;
        }
    **/
        @Override
        public boolean onOptionsItemSelected(MenuItem item) {
            // Handle action bar item clicks here. The action bar will
            // automatically handle clicks on the Home/Up button, so long
            // as you specify a parent activity in AndroidManifest.xml.
            int id = item.getItemId();

            //noinspection SimplifiableIfStatement
            if (id == R.id.action_settings) {
                return true;
            }

            return super.onOptionsItemSelected(item);
        }
    }
```

C. RL2_Reinforcement_strategy.py

```
    import numpy as np
    import random as rm
    import matplotlib.pyplot as plt
    from copy import copy
    from IPython.display import display, clear_output

    def printBoard(board):
        print('''
    bandwidth={} |Data={} |CPU_Instance={}
    -----
    app={} |Cloud_Vendor_Available={} |Location={}
    -----
    '''.format(*tuple(board)))

    def printBoardQs(board,Q):
        #printBoard(board)
        printParameters(board)
        Qs = [Q.get((tuple(board),m), 0) for m in range(3)]
        print('Reinforcements Received:')
        print('''Local Processing:{:.2f} | Offload on Local Servers:{:.2f} | Offload on
Remote Servers:{:.2f}
    '''.format(*Qs))


    def printParameters(board):
        print('''
    bandwidth= {} |Data= {} |CPU_Instance= {} |Wifi= {}
    '''.format(*tuple(board)))

    print('let\'s see what are my state parameters')
    #printBoard(np.array(['1','0','1','1','5','9']))
    printParameters(np.array(['Speed_Low','Data_Small','CPU_Low','On']))
```

```
    #print('okay now let\'s genearete a random number geneartor for each of these
parameters')

    #bandwidth = rm.randint(1,10)
    #bandwidth = np.random.randint(1,10,size=60)
    #data = np.random.randint(1,10, size = 60)
    #cpu = np.random.randint(1,10, size = 60)
    #app = np.random.randint(1,10, size = 60)
    #cloud_vendor = np.random.randint(1,10, size = 60)
    #location = rm.randint(1,10)
    #location = np.random.randint(1,10, size =60)

    Bandwidth = np.array(['Speed_Low','Speed_Normal','Speed_High'])
    Data = np.array(['Data_Small','Data_Medium','Data_Big'])
    CPU = np.array(['CPU_Low','CPU_Normal','CPU_High'])
    Wifi = np.array(['On','Off'])
    Out = np.array(['Local_Procssing','Offload_Local','Offload_Remote'])

    #print("location=",location)
    #board = np.array(['X',' ','O', ' ','X','O', 'X',' ',' '])
    #board1 = np.array([bandwidth,data,cpu,app,cloud_vendor,location])
    board2                                                                    =
np.array([rm.choice(Bandwidth),rm.choice(Data),rm.choice(CPU),rm.choice(Wifi)])
    print('print parameters')
    printParameters(board2)
    #print('print board1')
    #printBoard(board1)

    #Q = {} #empty table
    #Q[(tuple(board2),1)] = 4

    #print("Q:",Q)
    #print("Q[(tuple(board2),1)]:",Q[(tuple(board2),1)])
    #print("Q.get((tuple(board2),1),42):",Q.get((tuple(board2),1),42))

    #rho = 0.1 # learning rate
    #Q[(tuple(board),1)] += rho * (-1 - Q[(tuple(board),1)])
    #print("after  Q[(tuple(board),1)]  +=  rho  *  (-1  -  Q[(tuple(board),1)]):",
Q[(tuple(board),1)])
    #print('rm.choice(list(enumerate(Out))):',rm.choice(list(enumerate(Out))))
    #print('rm.choice(list(enumerate(Out)))[0]:',rm.choice(list(enumerate(Out)))[0])
    #print('list(enumerate(Out)):',list(enumerate(Out)))
    #print('list(Out):',list(Out))
    #print('Out:',Out)
    #print('list(enumerate(Out)):',list(enumerate(Out)))
    #print('list(enumerate(Out))[:0]:',list(enumerate(Out))[:0])
    #print('np.random.uniform():',np.random.uniform())
    #random_index = rm.randrange(0,len(Out))
    #print ('Out[random_index]:',Out[random_index])

    def epsilonGreedy(epsilon, Q, board, Out):
        #validMoves = np.where(board == ' ')[0]
        validMoves = np.array([0,1,2])
        #print('validMoves:',validMoves)
        if np.random.uniform() < epsilon:
            # Random Move
            tp = rm.choice(list(enumerate(Out)))[0]
            print('tp:',tp)
            return tp
            #return rm.choice(list(enumerate(Out)))[0]
            #return np.random.choice(validMoves)
        else:
            # Greedy Move
```

```
            Qs = np.array([Q.get((tuple(board),m), 0) for m in validMoves])
            tp = validMoves[ np.argmax(Qs) ]
            print('tp:',tp)
            return tp
            #return validMoves[ np.argmax(Qs) ]

    #print('epsilonGreedy(0.8,Q,board2,Out):',epsilonGreedy(0.8,Q,board2,Out))


    print('here goes part before for loop')

    maxGames = 200
    rho = 0.2
    epsilonDecayRate = 0.99
    epsilon = 0.8
    graphics = True
    showMoves = not graphics

    outcomes = np.zeros(maxGames)
    epsilons = np.zeros(maxGames)
    Q = {}

    if graphics:
        fig = plt.Figure(figsize=(10,10))

    print('here goes a for loop')
    #for i in range(60):
        #print (i)
        #location = np.random.randint(1,10, size =1)
    #    print("location=",location[i])
    #                                                                  board2                =
np.array([bandwidth[i],data[i],cpu[i],app[i],cloud_vendor[i],location[i]])
    #    printBoard(board2)
    #board2                                                                                    =
np.array([rm.choice(Bandwidth),rm.choice(Data),rm.choice(CPU),rm.choice(Wifi)])
    for nGames in range(maxGames):
        epsilon *= epsilonDecayRate
        epsilons[nGames] = epsilon
        step = 0
        move = epsilonGreedy(epsilon, Q, board2, Out)
        board2_all = {}
        board2                                                                                =
np.array([rm.choice(Bandwidth),rm.choice(Data),rm.choice(CPU),rm.choice(Wifi)])
        board2_all[nGames] = board2
        if (tuple(board2),move) not in Q:
                Q[(tuple(board2),move)] = 0  # initial Q value for new board,move

        if board2[3] == 'On':
            print('Wifi is ON')
            #if board2[0] == 'Speed_Low' and 'Speed_Normal':
            if board2[0] == 'Speed_Low' or board2[0] == 'Speed_Normal':
                print('Bandwidth = Speed_Low or Speed_Normal')

                if board2[1] == 'Data_Small' and board2[2] == 'CPU_High':
                    print('Data_Small and CPU_High so you can offload')
                    Q[(tuple(board2),1)] = 1
                    Q[(tuple(board2),2)] = 0
                    Q[(tuple(board2),0)] = -1
                else:
                    print('Don\'t offload')
                    Q[(tuple(board2),1)] = 0
                    Q[(tuple(board2),2)] = -1
                    Q[(tuple(board2),0)] = 1
```

```
            else:
                if board2[2] == 'CPU_Normal' or board2[2] == 'CPU_High':
                    print('CPU_Normal or CPU_High so you can offload')
                    Q[(tuple(board2),1)] = 1
                    Q[(tuple(board2),2)] = 0
                    Q[(tuple(board2),0)] = -1
                else:
                    print('Don\'t offload (this is second if loop)')
                    Q[(tuple(board2),1)] = 0
                    Q[(tuple(board2),2)] = -1
                    Q[(tuple(board2),0)] = 1
        else:
            print('Wifi is OFF')
            if board2[0] == 'Speed_Low' or board2[0] == 'Speed_Normal':
                print('Bandwidth = Speed_Low or Speed_Normal when wifi is off')
                if board2[1] == 'Data_Small' and board2[2] == 'CPU_High':
                    print('Data_Small and CPU_High so you can offload:Out2')
                    Q[(tuple(board2),1)] = 0
                    Q[(tuple(board2),2)] = 1
                    Q[(tuple(board2),0)] = -1
                else:
                    print('Don\'t offload when wifi is off')
                    Q[(tuple(board2),1)] = -1
                    Q[(tuple(board2),2)] = -1
                    Q[(tuple(board2),0)] = 1
            else:
                if board2[2] == 'CPU_Normal' or board2[2] == 'CPU_High':
                    print('CPU_Normal or CPU_High so you can offload:Out2')
                    Q[(tuple(board2),1)] = 0
                    Q[(tuple(board2),2)] = 1
                    Q[(tuple(board2),0)] = -1
                else:
                    print('Don\'t offload (this is second if loop when wifi is off)')
                    Q[(tuple(board2),1)] = -1
                    Q[(tuple(board2),2)] = -1
                    Q[(tuple(board2),0)] = 1

        #print (i)
        #location = np.random.randint(1,10, size =1)
    #    print("location=",location[i])
    #                                                          board2               =
np.array([bandwidth[i],data[i],cpu[i],app[i],cloud_vendor[i],location[i]])
    #    printBoard(board2)

    #-------------------------Just For Plotting the outcomes---------------
    print('after for loop')
    printBoardQs(board2,Q)

    outcomes = np.random.choice([-1,0,1],replace=True,size=(1000))
    #print('outcomes[:10]:',outcomes[:10])
    #print('Q:',Q)
    #print('Q.shape:',Q.shape) //did not work

    #print('Q.values():\n',Q.values())
    #print('Q.keys():\n\n',Q.keys())
    #print('Q.items():\n\n',Q.items())

    #for k in Q.keys():
    #  print(k, Q[k])

    #outcomes = np.array[Q.values()]
    #print('outcomes[:10]:',outcomes[:10])
```

```
names = ['id','data']
formats = ['f8','f8']
dtype = dict(names = names, formats=formats)
array=np.array([[key,val] for (key,val) in Q.iteritems()],dtype)
print(repr(array))
#plt.plot(Q)
def plotOutcomes(outcomes,epsilons,maxGames,nGames):
    if nGames==0:
        return
    nBins = 100
    nPer = int(maxGames/nBins)
    outcomeRows = outcomes.reshape((-1,nPer))
    outcomeRows = outcomeRows[:int(nGames/float(nPer))+1,:]
    avgs = np.mean(outcomeRows,axis=1)
    plt.subplot(3,1,1)
    xs = np.linspace(nPer,nGames,len(avgs))
    plt.plot(xs, avgs)
    plt.xlabel('Games')
    plt.ylabel('Mean of Outcomes (0=draw, 1=X win, -1=O win)')
    plt.title('Bins of {:d} Games'.format(nPer))
    plt.subplot(3,1,2)
    plt.plot(xs,np.sum(outcomeRows==-1,axis=1),'r-',label='Losses')
    plt.plot(xs,np.sum(outcomeRows==0,axis=1),'b-',label='Draws')
    plt.plot(xs,np.sum(outcomeRows==1,axis=1),'g-',label='Wins')
    plt.legend(loc="center")
    plt.ylabel('Number of Games in Bins of {:d}'.format(nPer))
    plt.subplot(3,1,3)
    plt.plot(epsilons[:nGames])
    plt.ylabel('$\epsilon$')



#plt.Figure(figsize=(8,8))
#plotOutcomes(outcomes,np.zeros(1000),1000,1000)
#plt.show()
#--------------------------Just For Plotting the outcomes---------------
```

D. RL3_RLstrategy.java

```
import numpy as np
import random as rm
import neuralnetworkQ as nn
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import copy



print( '\n---------------------------------------------------------------')
print( "Reinforcement Learning Example: Dynamic Marble on a Track")

# Define the problem

def reinforcement(s,sn):
    goal = 5
    return 0 if abs(sn[0]-goal) < 1 else -1
```

```
    def initialState():
        return np.array([10*np.random.random_sample(), 0.0])

    def nextState(s,a):
        s = copy.copy(s)   # s[0] is position, s[1] is velocity. a is -1, 0 or 1
        deltaT = 0.1                           # Euler integration time step
        s[0] += deltaT * s[1]                  # Update position
        s[1] += deltaT * (2 * a - 0.2 * s[1])  # Update velocity. Includes friction
        if s[0] < 0:        # Bound next position. If at limits, set velocity to 0.
            s = [0,0]
        elif s[0] > 10:
            s = [10,0]
        return s

    validActions = (-1,0,1)

    # training Loop
    gamma = 0.5
    nh = 5
    nTrials = 50
    nStepsPerTrial = 1000
    nSCGIterations = 10
    finalEpsilon = 0.01
    epsilonDecay = np.exp(np.log(finalEpsilon)/(nTrials)) # to produce this final value

    nnet = nn.NeuralNetworkQ(3,nh,1,((0,10), (-3,3), (-1,1)))
    epsilon = 1
    epsilonTrace = np.zeros(nTrials)
    rtrace = np.zeros(nTrials)
    for trial in range(nTrials):
        # Collect nStepsPerRep samples of X, R, Qn, and Q, and update epsilon
        X,R,Qn,Q,epsilon = nnet.makeSamples(initialState,nextState,reinforcement,
                                            validActions,nStepsPerTrial,epsilon)
        # Update the Q neural network.
        nnet.train(X,R,Qn,Q,gamma=gamma,          nIterations=nSCGIterations)         #
weightPrecision=1e-8, errorPrecision=1e-10)
        epsilon *= epsilonDecay
        # Rest is for plotting
        epsilonTrace[trial] = epsilon
        rtrace[trial] = np.mean(R)

        print('Trial',trial,'mean R',np.mean(R))


    ##  Plotting functions

    def plotStatus(net,trial,epsilonTrace,rtrace):
        plt.subplot(4,3,1)
        plt.plot(epsilonTrace[:trial+1])
        plt.ylabel("Random Action Probability ($\epsilon$)")
        plt.ylim(0,1)
        plt.subplot(4,3,2)
        plt.plot(X[:,0])
        plt.plot([0,X.shape[0]], [5,5],'--',alpha=0.5,lw=5)
        plt.ylabel("$x$")
        plt.ylim(-1,11)
        #qs = [[net.use([s,0,a]) for a in actions] for s in range(11)]
        qs = net.use(np.array([[s,0,a] for a in validActions for s in range(11)]))
        #print np.hstack((qs,-1+np.argmax(qs,axis=1).reshape((-1,1))))
        plt.subplot(4,3,3)
        acts = ["L","0","R"]
        actsiByState = np.argmax(qs.reshape((len(validActions),-1)),axis=0)
```

```
        for i in range(11):
            plt.text(i,0,acts[actsiByState[i]])
            plt.xlim(-1,11)
            plt.ylim(-1,1)
        plt.text(2,0.2,"Policy for Zero Velocity")
        plt.axis("off")
        plt.subplot(4,3,4)
        plt.plot(rtrace[:trial+1],alpha=0.5)
        #plt.plot(np.convolve(rtrace[:trial+1],np.array([0.02]*50),mode='valid'))
        binSize = 20
        if trial+1 > binSize:
            # Calculate mean of every bin of binSize reinforcement values
            smoothed                                                                    =
np.mean(rtrace[:int(trial/binSize)*binSize].reshape((int(trial/binSize),binSize)),axis
=1)
            plt.plot(np.arange(1,1+int(trial/binSize))*binSize,smoothed)
        plt.ylabel("Mean reinforcement")
        plt.subplot(4,3,5)
        plt.plot(X[:,0],X[:,1])
        plt.plot(X[0,0],X[0,1],'o')
        plt.xlabel("$x$")
        plt.ylabel("$\dot{x}$")
        plt.fill_between([4,6],[-5,-5],[5,5],color="red",alpha=0.3)
        plt.xlim(-1,11)
        plt.ylim(-5,5)
        plt.subplot(4,3,6)
        net.draw(["$x$","$\dot{x}$","$a$"],["Q"])

        plt.subplot(4,3,7)
        n = 20
        positions = np.linspace(0,10,n)
        velocities =  np.linspace(-5,5,n)
        xs,ys = np.meshgrid(positions,velocities)
        #states = np.vstack((xs.flat,ys.flat)).T
        #qs = [net.use(np.hstack((states,np.ones((states.shape[0],1))*act))) for act in
actions]
        xsflat = xs.flat
        ysflat = ys.flat
        qs = net.use(np.array([[xsflat[i],ysflat[i],a] for a in validActions for i in
range(len(xsflat))]))
        #qs = np.array(qs).squeeze().T
        qs = qs.reshape((len(validActions),-1)).T
        qsmax = np.max(qs,axis=1).reshape(xs.shape)
        cs = plt.contourf(xs,ys,qsmax)
        plt.colorbar(cs)
        plt.xlabel("$x$")
        plt.ylabel("$\dot{x}$")
        plt.title("Max Q")
        plt.subplot(4,3,8)
        acts = np.array(validActions)[np.argmax(qs,axis=1)].reshape(xs.shape)
        cs = plt.contourf(xs,ys,acts,[-2, -0.5, 0.5, 2])
        plt.colorbar(cs)
        plt.xlabel("$x$")
        plt.ylabel("$\dot{x}$")
        plt.title("Actions")

        s = plt.subplot(4,3,10)
        rect = s.get_position()
        ax = Axes3D(plt.gcf(),rect=rect)
        ax.plot_surface(xs,ys,qsmax,cstride=1,rstride=1,cmap=cm.jet,linewidth=0)
        ax.set_xlabel("$x$")
        ax.set_ylabel("$\dot{x}$")
        #ax.set_zlabel("Max Q")
```

```
    plt.title("Max Q")

    s = plt.subplot(4,3,11)
    rect = s.get_position()
    ax = Axes3D(plt.gcf(),rect=rect)
    ax.plot_surface(xs,ys,acts,cstride=1,rstride=1,cmap=cm.jet,linewidth=0)
    ax.set_xlabel("$x$")
    ax.set_ylabel("$\dot{x}$")
    #ax.set_zlabel("Action")
    plt.title("Action")

def testIt(Qnet,nTrials,nStepsPerTrial):
    xs = np.linspace(0,10,nTrials)
    plt.subplot(4,3,12)
    for x in xs:
        s = [x,0] ## 0 velocity
        xtrace = np.zeros((nStepsPerTrial,2))
        for step in range(nStepsPerTrial):
            a,_ = Qnet.epsilonGreedy(s,validActions,0.0) # epsilon = 0
            s = nextState(s,a)
            xtrace[step,:] = s
        plt.plot(xtrace[:,0],xtrace[:,1])
        plt.xlim(-1,11)
        plt.ylim(-5,5)
        plt.plot([5,5],[-5,5],'--',alpha=0.5,lw=5)
        plt.ylabel('$\dot{x}$')
        plt.xlabel('$x$')
        plt.title('State Trajectories for $\epsilon=0$')



plotStatus(nnet,nTrials,epsilonTrace,rtrace)
testIt(nnet,10,500)

plt.show()
```

E. A5. Mobile-AWS cloud interaction -1

Upload to S3

Here is the code we use to upload the picture files:

```
   def push_picture_to_s3(id):
     try:
       import boto
       from offloading.s3.key import Key
       # set offloading lib debug to critical
       logging.getLogger('offloading').setLevel(logging.CRITICAL)
       bucket_name = settings.MyCloudBucketOffloading
       # connect to the bucket
       conn = boto.connect_s3(settings.AWS_ACCESS_KEY_ID,
                       settings.AWS_SECRET_ACCESS_KEY)
       bucket = conn.get_bucket(bucket_name)
       # go through each version of the file
       key = '%s.png' % id
       fn = '/var/www/data/%s.png' % id
       # create a key to keep track of our file in the storage
       k = Key(bucket)
       k.key = key
```

```
        k.set_contents_from_filename(fn)
        # we need to make it public so it can be accessed publicly
        # using a URL like http://s3.amazonaws.com/bucket_name/key
        k.make_public()
        # remove the file from the web server
        os.remove(fn)
      except:
```

Download from S3

We can access the file using the URL: http://s3.amazonaws.com/bucket_name/key

Here is the script to do that:

```
import boto
import sys, os
from offloading.s3.key import Key

LOCAL_PATH = '/backup/s3/'
AWS_ACCESS_KEY_ID = some_key
AWS_SECRET_ACCESS_KEY = some_secret_key

bucket_name = 'MyCloudBucketOffloading'
# connect to the bucket
conn = Offloading.connect_s3(AWS_ACCESS_KEY_ID,
                AWS_SECRET_ACCESS_KEY)
bucket = conn.get_bucket(bucket_name)
# go through the list of files
bucket_list = bucket.list()
for l in bucket_list:
  keyString = str(l.key)
  # check if file exists locally, if not: download it
  if not os.path.exists(LOCAL_PATH+keyString):
    l.get_contents_to_filename(LOCAL_PATH+keyString)
```

F. A6. Mobile-AWS cloud interaction - 2

```
aws s3 mb s3://MyCloudBucketOffloading //  create a bucket on AWS cloud
aws s3 cp stuff/firstfile.txt s3://MyCloudBucketOffloading  // upload the file on AWS cloud
aws s3 ls s3://MyCloudBucketOffloading // see all the file which are present in Bucket
aws s3 sync . s3://MyCloudBucketOffloading/stuff – - delete //sync files on cloud bucket
aws s3 rb s3://MyCloudBucketOffloading - - force // delete the bucket
```

G. A7. LDAStrategy.java

```
//-------------------------------------------------------------------------
// Author: Aditya Khune
//
// Description   : Describes the LDA algorithm based on Dr Charles Anderson's code
// in CS 545, 2014
//
//-------------------------------------------------------------------------
```

```
import java.util.*;
import java.awt.*;

/**
 * Implements the Linear Discriminant Analysis Algorithm
 */
public class AlgorithmLDA extends Algorithm
{
    // Public Data Members
    //
    Vector<MyPoint> decision_regions_d;
    Vector<MyPoint> support_vectors_d;
    int output_canvas_d[][];

    Matrix W;
    Matrix LDA;
    Matrix CLDA;
    Matrix B;
    Matrix S;
    Matrix invW;

   /**

    * @return   Returns true.
    */
    public boolean initialize()
    {
    // algo_id = "AlgorithmLDA";

    // Debug
    //
    // System.out.println(algo_id + " initialize()");

    step_count = 2;
    point_means_d      = new Vector<MyPoint>();

    description_d      = new Vector<String>();

    // Initialize local Matrix objects
    //
    W    = new Matrix();
    LDA  = new Matrix();
    CLDA = new Matrix();
    invW = new Matrix();

    // Add the process description for the LDA algorithm
    //
    if (description_d.size() == 0)
        {
        String str = new String("   0. Initialize the original data.");
        description_d.addElement(str);


        str = new String("   2. Computing the means and covariance.");
        description_d.addElement(str);

        str = new String("   3. Computing the decision regions based on the class
independent LDA algorithm.");
        description_d.addElement(str);
    }
```

```
// append message to process box
//
pro_box_d.appendMessage("Class Independent LDA Analysis:" + "\n");

// set the data points for this algorithm
//
//  set1_d = (Vector<MyPoint>)data_points_d.dset1.clone();
//  set2_d = (Vector)data_points_d.dset2.clone();


set1_d = data_points_d.dset1;
set2_d = data_points_d.dset2;



// set the step index
//
step_index_d = 0;

// append message to process box
//
pro_box_d.appendMessage((String)description_d.get(step_index_d));

// exit initialize
//
return true;
}
```

ABBREVATIONS

3D	3-Dimensional
AWS	Amazon Web Service
AWS S3	Amazon Web Services Simple Storage Service
AWS EC2	Amazon Web Services Elastic Cloud Compute
RL	Reinforcement Learning
GPS	Global Positioning System
CPU	Central Processing Unit
ADB	Android Device Bridge
SDK	Software Development Kit
Wi-Fi	Wireless Fidelity
2G	Second-generation wireless telephone technology
3G	Third-generation wireless telephone technology
HSPA	High Speed Packet Access
HSPA+	Evolved HSPA or 4G
4G	Fourth-generation wireless telephone technology

OC	Offloading Candidate
VM	Virtual Machine