THESIS

AN INTELLIGENT, MOBILE NETWORK AWARE MIDDLEWARE FRAMEWORK FOR ENERGY EFFIECIENT 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

ABSTRACT

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

The users of smartphones demand longer battery life today. Low-power software and hardware design has been an active research topic for many years. Offloading applications on a surrogate machine is an innovative technique in this area that is being explored for saving energy and increasing responsiveness of mobile devices. Offloading simply means transferring the computation or data on the cloud. Offloading has generated a renewed interest in smartphone research community due to advancement in cloud computing and high-speed mobile networks such as 4G,

Cloud computing is a new paradigm in which computing resources such as processing, memory, and storage are not physically present at the user's location. Instead, a service provider owns and manages these resources, and users access them via the Internet.

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 to highlight its real potential in real time smartphone applications. The effect of varying network technologies is also a major concern in this area.

In this work, we study some of the widely used smartphone apps in both local and offload processing modes. Our results are helpful to identify the advantages and disadvantages of offloading with varying mobile networks. Further, we have presented a Machine Learning based strategy to enhance the offloading system. Our strategy helps smartphone make decision of choosing between available networks (3G, 4G or Wi-Fi) while offloading mode is active. The proposed system considers suitable information on the device to make accurate offloading decision in order to get optimized energy consumption.

In our strategy, we propose an application and user interaction aware middleware framework that uses machine learning techniques such as Reinforcement Learning and Fuzzy Logic methods for making offloading decisions effectively. We have also used Neural Network to test our reward based mechanism in an simulated environment. We have analyzed the validity of our algorithm with the help of compute intensive benchmark applications. Together these techniques allow minimization of energy consumption in mobile offloading systems.

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. Sangmi Pallickara for serving as my committee members. I would also like to thank my colleagues in Multicore Embedded Systems (MECS) lab for their advice and unwavering support. I am grateful to my friend 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 the least, I would like to thank my family: my loving parents Dilip Khune, Geeta Khune, and my sister Apurva for the support they provided me through my entire life. Your prayer for me was what sustained me thus far.

DEDICATION

To my parents, sister

and

To the Devotees of Krishna in Colorado

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

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Faster network speeds and rapid innovations in mobile technologies have changed the way we used our computers. Thanks to new operating system architectures such as Android and iOS, the number of applications on our smartphones have literally exploded. Figure 1.1 shows the number of apps available in the leading app stores until 2015. Smartphones are widely used for navigating numerous important life activities, from monitoring a health condition to accessing educational resources. Unfortunately, most of the time smartphone users face an annoying situation where they have to recharge their handsets twice per day.

Today's smartphones offer variety of complex applications, larger communication bandwidth and more processing power. However, this has increased the burden on its energy usage, while it is seen that advances in battery capacity do not keep up with the requirements of the modern user.

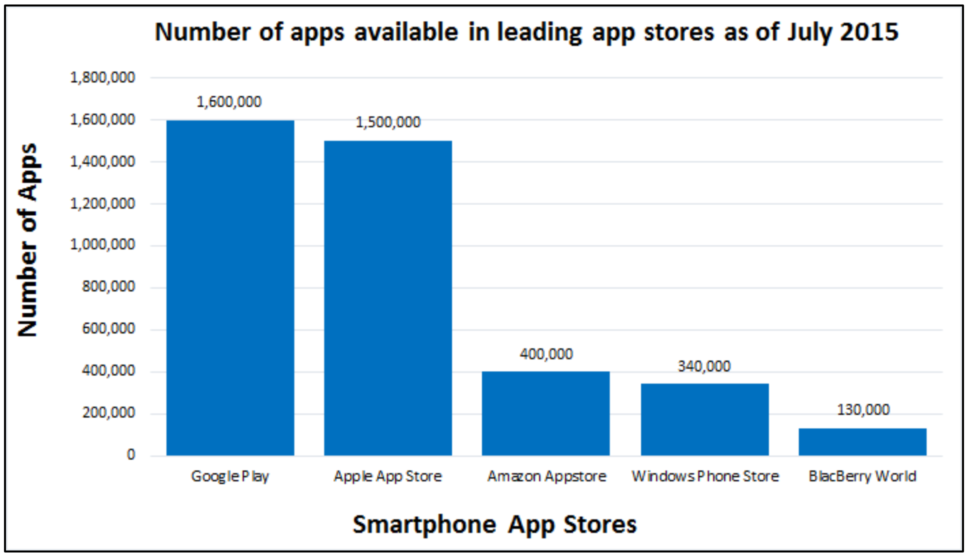


Figure 1.1 Number of Smartphone apps available in App Stores [22]

Cloud computing has drawn attention of mobile technologies due to the increasing demand of applications, for processing power, storage place, and energy. Cloud computing promises availability of infinite resources and it mainly operates with utility computing model, where consumers pay based on their usage. Today, 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.

Offloading mobile computation on cloud is being widely considered for saving energy and increasing responsiveness of mobile devices. The potential of code offloading lies in the ability to sustain power hungry applications by identifying and managing energy consuming resources of the mobile device by offloading them on cloud.

Currently the research in this area is focused on providing the device with an offloading logic, based on its local context. In this work, we have done application-oriented study of offloading in smartphones. We conducted a set of experiments using smartphone offloading applications with available networks (3G, 4G (HSPA+) and Wi-Fi). Finally, we propose a Reinforcement Learning (RL) based system that will help smartphone choose the right network to do offloading in order to reduce battery consumption.

1.1 Can offloading computation save energy?

To answer this question Kumar et al. provide a mathematical analysis of offloading in [2]. 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 bandwidth (B). If (D/C) is low, then offloading can save energy.

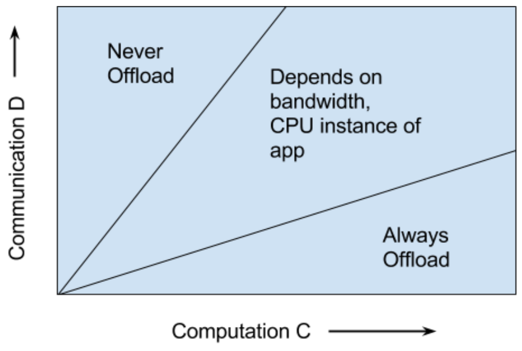


Figure 1.2 : Offloading is beneficial when large amounts of computation C are needed with relatively small amounts of communication D [2]

In simplified words, offloading is preferable only if a mobile operation requires high amounts of computational processing and at the same time, low amounts of data need to be sent in the communication.

Existing studies thus focus on determining whether to offload computation by predicting the relationships among these three factors. Multiple research works have proposed different strategies to empower mobile devices with intelligent offloading system. We can see a fundamental offloading architecture in the Figure 1.3. Offloading relies on remote servers to execute code delegated by a mobile device.

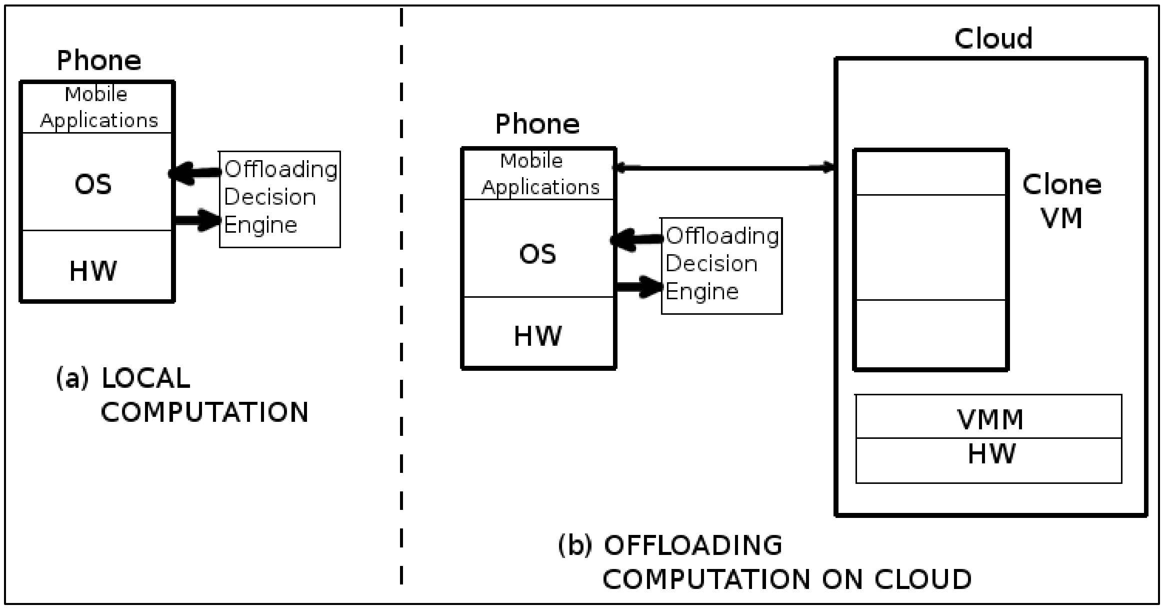


Figure 1.3: Offloading System Model

In the architecture that is presented in the paper 'Mobile code offloading: from concept to practice and beyond' [3], 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.

System profilers monitor multiple parameters of the smartphone, such as available bandwidth, data size to transmit, and energy to execute the code. We look to these parameters to know *when to offload* to the cloud.

The code profiler determines *what to offload* (portions of code: Method, Thread, or Class). Code partitioning requires the selection of the code to be offloaded. Code can be partitioned through different strategies for instance, in the paper presented by Cuervo et al. [4], the static annotations are used by a software developer to select the code that should be offloaded. In the paper 'Clonecloud: elastic execution between mobile device and cloud' [5], Chun et al. have presented an automated mechanism, which analyzes the code during runtime. Automated mechanisms are preferable over static ones as they can adapt the code to be executed in different devices.

The decision engine analyzes the parameters from System and code profilers and applies certain logic over them to deduce *when to offload*. If the engine concludes a positive outcome, the offloading system is activated, which sends the required data and the code is invoked remotely on the cloud; otherwise, the processing is performed locally.

1.2 Challenges in computation offloading

The offloading technique is far from being adopted in the design of current mobile architectures; this is because utilization of code offloading in real scenarios proves to be mostly negative [6], which means that the device spends more energy in the offloading process than the actual energy that is saved. In this section, we highlight the challenges and obstacles in deploying code offloading.

We found out that network inconsistency is one of the major concerns while we want to make a correct offloading decision to offload the processing on cloud. In Chapter 5, we have discussed more about this problem of network inconsistency.

It is very difficult to evaluate runtime properties of code, the code will have non-deterministic behavior during runtime, it is difficult to estimate the running cost of a piece of code considered for offloading [3]. The code in consideration of offloading might become intensive based on factors such as the user input, type of application, execution environment, available memory, etc. [6].

Code partitioning is one of the mechanisms considered by researchers. Code partitioning relies on the expertise of the software developer; the main idea is to annotate portions of code statically. These annotations can cause poor flexibility to execute the app in different mobile devices; it can cause unnecessary code offloading that drains energy. Automated strategies are shown to be ineffective, and need a major low-level modification in the core system of the mobile platform, which lead to privacy issues [3].

The offloading decision engine in the mobile device should consider not only the potential energy savings, but also the response time of the request. It can also be argued that, as the computational capabilities of the latest smartphones are comparable to some servers running in the cloud, in such case why to offload. In this work, we have addressed some of these challenges by using Reinforcement Learning (RL) decision engine for offloading process.

1.3 Contributions

- In this thesis, we investigate the viability of offloading solutions in detail with the help of benchmark applications to know right kind of applications that may benefit from effective offloading Architecture. During the evaluation, we run these applications using all available networks (3G, 4G, and Wi-Fi).
- In this work, we present a novel adaptive offloading technique that uses Reinforcement Learning (RL) and Fuzzy Logic to deduce right offloading decision. The decision is taken so that offloading is guaranteed to optimize both the response time and energy consumption. This method is classified as a 'Unsupervised Learning' method. We have also

used Neural Network to test our reward based machine learning mechanism in an simulated environment.

- Further, we used supervised learning methods such as Linear Discriminant Analysis to create the offloading decision engines. Finally we compare our proposed techniques with prior work in the offloading domain that propose to empower decision engines with offloading logic for instance decision engine with SVR, Fuzzy Logic and other supervised learning techniques.

1.4 Outline

The rest of the thesis is organized as follows. Chapter 2 gives a detailed overview of prior works in energy saving and offloading in smartphones. In Chapter 3, we describe the machine learning techniques used in this thesis to create an offloading decision engine. In Chapter 4, we present a comprehensive real-time application oriented study of offloading using available networks such as 3G, 4G and Wi-Fi. Further, we have selected some of the popular applications for our experimentations on a smartphone and presented our findings based on the results. In Chapter 5, we propose our methods based on machine learning techniques such as Reinforcement Learning and Fuzzy Logic; and we suggest the need of smart offloading decision mechanisms to choose between available networks and counter network inconsistency. In the final sections of this chapter, we have compared our results with important prior works.

Chapter 6 concludes the thesis with a summary and future work. The appendix offers the source code of the strategies presented in our thesis.

CHAPTER 2

LITERATURE REVIEW

A substantial amount of work has been done in the area of smartphone offloading for mobile devices in recent years. The advances in smartphone processing and storage technology outpaces the advances in the battery technology. Computation offloading has been seen as a potential solution for the smartphone's energy bottleneck problem. In this Chapter, we give an overview of the research in computation offloading in order to increase energy efficiency of Smartphones.

In [4] authors have proposed a system called MAUI, it has a strategy based on code annotations to determine which methods from a Class must be offloaded. Annotations are introduced within the source code by the developer at development phase itself. At runtime, methods are identified by the MAUI profiler, which performs the offloading over the methods, if bandwidth of the network and data transfer conditions are ideal. MAUI optimizes both the energy consumption and execution time using an optimization solver. However, this annotation method puts extra burden on the side of already complex software development phase. We have proposed an automated machine learning approach in this thesis.

In [5] authors have proposed CloneCloud, which is a system for elastic execution between mobile and cloud through dynamic application partitioning, where a thread of the application is migrated to a clone of the smartphone in the cloud. CloneCloud uses dynamic profiling and optimization solver. In CloneCloud partitioning takes place without the developer intervention and application partitioning is based on static analysis to specify the migration and reintegration points in the application.

In [2] authors have proposed an equation with several parameters to measure whether computation offloading to cloud would save energy or not. These parameters are in this paper, the authors discussed various parameters such as network bandwidth, cloud-processing speed, device processing speed, the number of transferred bytes, and the energy consumption of a smartphone when it is in idle, processing and communicating states. However, we found that authors did not experiment their work in a real offloading framework.

In paper [7] authors proposed ThinkAir which is a computation offloading system that is similar to MAUI and Cloudclone. ThinkAir does not only focus on the offloading efficiency but also on the elasticity and scalability of the cloud side; it boosts the power of mobile cloud computing through parallelizing method execution using multiple virtual machine images. In papers [8], [9] and [10] authors have concentrated on adaptive learning and proposed offloading decision engines based on machine learning techniques. We have studied some of their strategies in later part of this thesis so that we can compare our methods.

'Smartphone Energizer' [9] is a supervised learning-based technique for energy efficient computation offloading. In this work, authors propose an adaptive, and context-aware offloading technique that uses Support Vector Regression (SVR) and several contextual features to predict the remote execution time and energy consumption then takes the offloading decision. The decision is taken so that offloading is guaranteed to optimize both the response time and energy consumption. Smartphone Energizer Client (SEC) initially starts in the learning mode, in which it extracts the different network, device, and application features and stores them after each service invocation. After each local service invocation, the Smartphone Energizers profiler stores the context parameters, which include the service identifier, input size, and output size along with the consumed energy and time during service execution. When the number of local service invocations exceeds the local learning capacity, the SEC switches to the remote execution by checking if there's a reachable offloading server, then the service will be installed on the server (for the first time only) and will be executed remotely, otherwise it will be executed locally.

In [8] the authors have proposed fuzzy decision engine for code offloading, that considers both mobile and cloud variables. At the mobile platform level, the device uses a decision engine based on fuzzy logic, which is utilized to combine n number of variables, which are to be obtained from the overall mobile cloud architecture. Fuzzy Logic Decision Engine works in three steps namely: Fuzzyfication, Inference and Defuzzification. The distribution of different technologies around the world varies significantly. But there is a serious problem with this approach when we 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. It is difficult to rely on the fuzzy logic decision engine presented in [8] because the app developers will have to customize the decision engines depending upon which part of the world the device lies. We found that our method based on Reinforcement Learning could deal with this problem effectively.

CHAPTER 3

OVERVIEW OF MACHINE LEARNING TECHNIQUES

Machine learning algorithms search for patterns and regularities in any given data and have found wide usage across various application domains. They automatically learn from data by generalizing from examples. As more data becomes available, problems that are more ambitious can be tackled. These algorithms are typically implemented in two phases. In the first phase, called ***training phase***, data is gathered and provided to the algorithm, so it can learn patterns and create a model to classify data or predict data properties. In the second phase, called ***testing phase***, new data is tested against the model that was built during the training phase, and the effectiveness of the model is revealed. Such two-phase learning algorithms are called *supervised* learning algorithms. There are also algorithms in which the testing phase is not used, and such algorithms are called *unsupervised* learning algorithms. These algorithms use unlabeled data to cluster the data in different classes. Machine learning algorithms can be used for classification or regression. In *classification,* the machine-learning algorithm learns to classify the data in different classes while in *regression* it predicts a continuous variable by learning from the train data. To improve offloading decision accuracy over prior work, we propose to integrate machine-learning techniques that intelligently make use of different modules in our offloading framework.

3.1 Reinforcement Learning (RL)

Reinforcement learning is learning by interacting with an environment. An RL agent learns from the consequences of its actions, rather than from being explicitly taught and it selects its actions on basis of its past experiences (exploitation) and also by new choices (exploration), which is essentially trial and error learning. The reinforcement signal that the RL-agent receives is a numerical reward, which encodes the success of an action's outcome, and the agent seeks to learn to select actions that maximize the accumulated reward over time.

A reinforcement-learning engine interacts with its environment in discrete time steps. At each time t, the agent receives an observation *Ot*, which typically includes the reward rt. It then chooses an action at from the set of actions available, which is subsequently sent to the environment. The environment moves to a new state s_{t+1} and the reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is determined. The goal of a reinforcement-learning agent is to collect as much reward as possible. The agent can choose any action as a function of the history and it can even randomize its action selection.

Reinforcement learning is particularly well suited to problems, which include a long-term versus short-term reward tradeoff. It has been applied successfully to various problems, including robot control, elevator scheduling, telecommunications, etc.

The basic reinforcement-learning model consists of:

- a set of environment states S;
- a set of actions A;
- rules of transitioning between states;
- rules that determine the scalar immediate reward of a transition

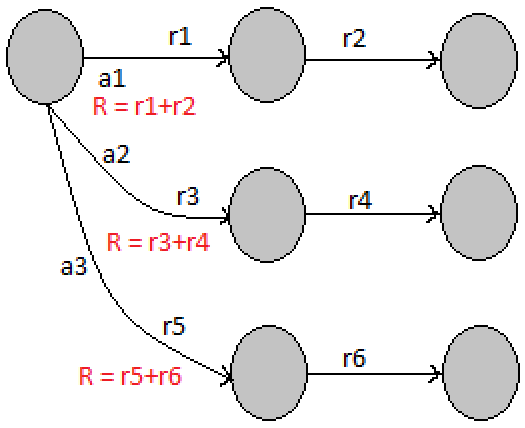


Figure 3.1 : Choose from available actions with best Reinforcement History

We want to choose the action that we predict will result in the best possible future from the current state in Figure 3.1. Need a value that represents the future outcome. With the correct values, multi-step decision problems are reduced to single-step decision problems. Just pick action with best value and it has guaranteed to find optimal multi-step solution! The utility or cost of a single action taken from a state is the reinforcement for that action from that state. The value of that state-action is the expected value of the full return or the sum of reinforcements that will follow when that action is taken.

Say we are in state s_t at time t. Upon taking action at from that state we observe the one-step reinforcement r_{t+1}, and the next state s_{t+1}. Say this continues until we reach a goal state, K steps later we have return as:

$$R_t = \sum_{k=0}^{K} r_{t+k+1} \tag{3.1}$$

So the aim of Reinforcement Learning algorithm generally is either to maximize or minimize the reinforcements R_t depending upon our Reinforcement function.

3.1.1 Q Function

A function called Q function stores the reinforcement values for each case it encounters, some more mathematics about this Q function which is used in RL algorithm is shown here:

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. We will call the state-action value function Q.

$$Q(s_t, a_t) \approx \sum_{k=0}^{\infty} r_{t+k+1} \tag{3.2}$$

Here, s_t = state, a_t = actions, and r_t = reinforcements received.

3.1.2 Reinforcement Learning Objective

The objective for any reinforcement learning problem is to find the sequence of actions that maximizes (or minimizes) the sum of reinforcements along the sequence. This is reduced to the objective of acquiring the Q function, which predicts the expected sum of future reinforcements, because the correct Q function determines the optimal next action.

Therefore, the RL objective is to make this approximation as accurate as possible:

$$\text{Minimize E}\left(\sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t)\right)^2 \tag{3.3}$$

This is usually formulated as the least squares objective:

(3.4)

3.2 RL with Neural Networks

Neural network models, also known as *Artificial Neural Networks*, are inspired by the way the human brain is believed to function. Many of the normal basic everyday information processing requirements handled by the brain, for example sensory processing, cognition, and learning, surpass any capable computing system out there today. Although a human brain is quite different than today's computing hardware, it is believed that the basic concepts still apply in that there is a computational unit, known as a *neuron*, and connections to memory stored in *synapses*. The main difference being that the human brain consists of billions of these simple parallel processing units, neurons, which are interconnected in a massive multi-layered distributive network of synapses and

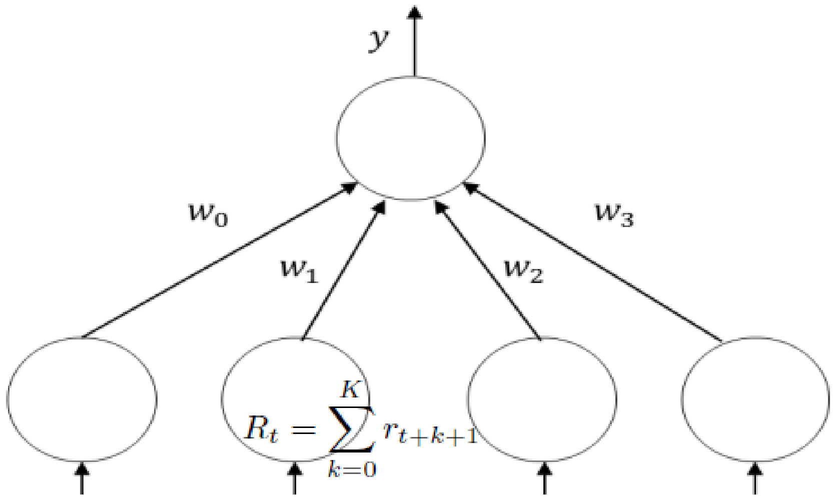


Figure 3.2: Neural network perceptron model

neurons. In machine learning, these concepts are modeled as what is called a *perceptron*, the basic processing element, connected by other *perceptrons* through weighted connections, as illustrated in figure 5.3. The output of a perceptron is simply a weighted sum of its inputs including a weighted bias, as shown in following equation.

$$y = \sum_{i=1}^{n} w_i x_i + w_0 \tag{3.5}$$

To compute the output y given a sample x_i, *backpropagation* using the gradient with respect to the weights is performed using a training dataset to find the weight parameters, w_i, that minimize the mean squared error between the neural network outputs, y_i, and the target outputs, t_i. By default, the neural network consists of a hyperplane (for multiple perceptrons) that can be used as a linear discriminant to linearly separate the classes. To improve prediction accuracy, we make it non-linear, by applying a sigmoidal or hyperbolic tangent to hidden unit layer perceptrons, as denoted in equation below. This allows for non-linear boundaries with the output of the neural network being linear in the weights, but non-linear in the inputs.

$$y_i' = sigmoid(y_i) = \frac{1}{1 + \exp(\boldsymbol{w^T x})} \tag{3.6}$$

For classification with a neural network (non-linear logistic regression), the number of parallel output perceptrons is equal to the number of classes. The output from each perceptron, y_i, is then sent to post processing as in equation below to determine the respective class by taking the maximum of the post-processed outputs:

$$C_i \; if \; y_i = \max_i \frac{\exp(y_i)}{\sum_i \exp(y_i)} \tag{3.7}$$

One of the biggest criticisms about the use of neural networks is the time required for training. Although this can be a major issue if using a simple gradient descent approach, newer training techniques, such as the scaled conjugate gradient (SCG) [21], can greatly minimize the time required for training. SCG, a method for efficiently training feed-forward neural networks, was used for training the neural networks in this thesis.

3.3 Fuzzy Logic

Fuzzy Logic deals with approximate rather than fixed reasoning, and it has capabilities to react to continuous changes of the dependent variables. The decision of offloading processing components to cloud becomes a variable, and controlling this variable can be a complex task due to many real-time constraints of the overall mobile and cloud system parameters.

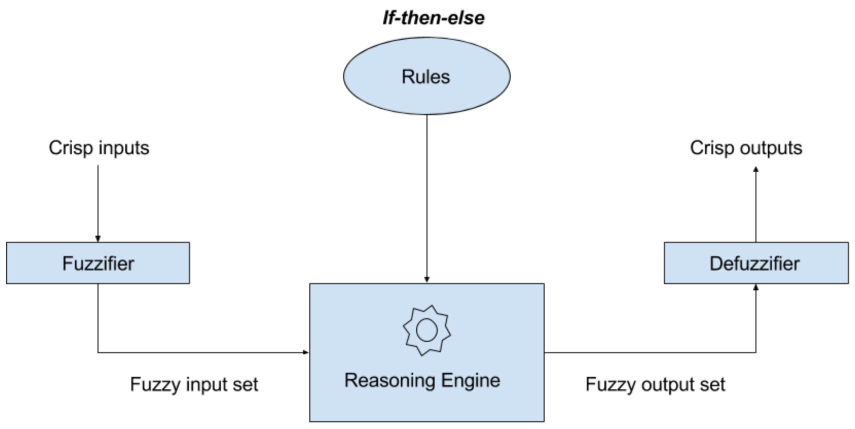


Figure 3.3 Fuzzy Logic Decision Engine [8]

Fuzzy Logic Decision Engine works in three steps namely: Fuzzyfication, Inference and Defuzzification. Let us see these steps in detail: 1) In Fuzzification, input data is converted into linguistic variables, which are assigned to a specific membership function. 2) A reasoning engine is applied to the variables, which makes an inference based on a set of rules. Finally, 3) The output from reasoning engine are mapped to linguistic variable sets again (aka defuzzyification).

3.4 Linear Discriminant Analysis (LDA)

In machine learning, classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Classification is considered an instance of supervised learning, i.e. learning where a training set of correctly identified observations is available. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. Linear discriminant analysis (LDA) makes use of a Bayesian approach to classification in which parameters are considered as random variables of a prior distribution. This concept is fundamentally different from data-driven linear and non-linear discriminant analyses in which what is learned is a function that maps or separates samples to a class. Bayesian estimation and the application of LDA is also known as generative modeling.

CHAPTER 4

APPLICATION ORIENTED STUDY OF OFFLOADING

We surveyed various applications that are likely to benefit from offloading as suggested by important publications in this area. Here is a list of types of applications as follows: applications with 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. These applications consume large mobile battery, memory, and computational resources.

In this chapter, we analyze the performance implications of offloading solutions by comparing the two extreme points – one that does not use the cloud at all, and another that primarily relies on the cloud. We have selected out five popular commercially available smartphone applications for various user operations as follows:

- Matrix Operations
- Internet Browsers
- Zipper
- Voice Recognition and Translation
- Torrents

Our evaluation focuses on two metrics: (i) battery consumption, and (ii) response time. We have compared the results obtained with each of available network 3G, 4G (HSPA+) and Wi-Fi in order to know which is the best possible option for offloading data/processing on cloud. We consider this metric to be relevant for the foreseeable future. To decrease the interference of the screen while doing energy analysis we run the applications with minimum brightness. Power consumption is measured by monsoon power analysis tool.

4.1 Experimental Setup

The power estimation models were built using real power measurements on the Samsung S3 device. The contact between the smartphone and the battery was instrumented, and current was measured using the Monsoon Solutions power monitor [20]. The monitor connects to a PC running the Monsoon Solutions power tool software that allows real-time current/power measurements over time. The power monitor setup is shown in figure 4.1.

We have run each experiment 15 times on each handsets mentioned above, and then averaged out the readings obtained. All the experiments are done using AT & T's 3G, 4G (HSPA+) Network and Comcast's Wi-Fi Network in west Fort Collins area. We run a majority of our experiments during the night time when the load in the cell tower is low.

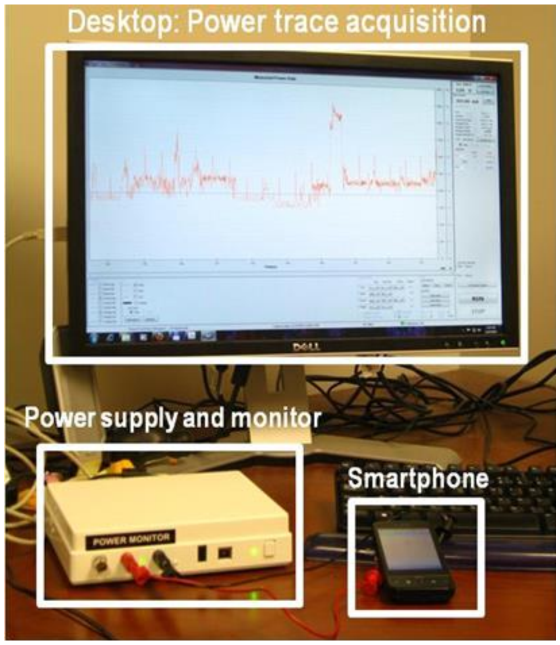


Figure 4.1 Power Monitor Setup

4.2 Smartphone Applications

In this section, we present the results obtained after the experimentations on smartphone applications. To keep the focus on the scientific aspects of our study, we anonymize these applications and compare traditional locally run computation applications with the cloud based applications.

4.2.1 Matrix Operations

Numerous applications involve some kind of large or small matrix operations, for instance image-processing applications. Many matrix calculator apps are available which are popular in android smartphones to do matrix calculations. In this experiment, we have chosen one such app; this application calculates values of Inverse of a matrix.

In Figure 4.2, we can see the battery consumption of smartphone increases manifolds as the size

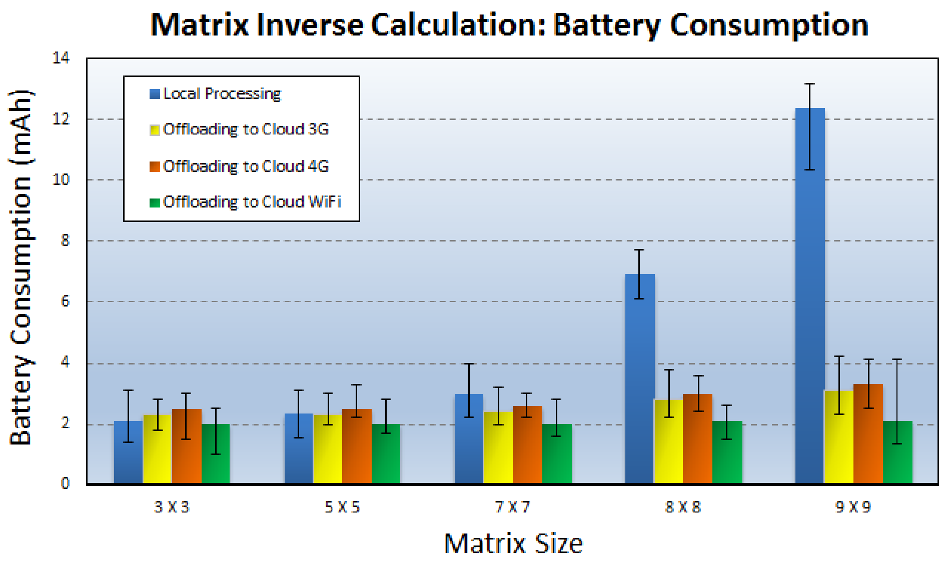


Figure 4.2: Battery Consumption for Matrix Inverse Calculation

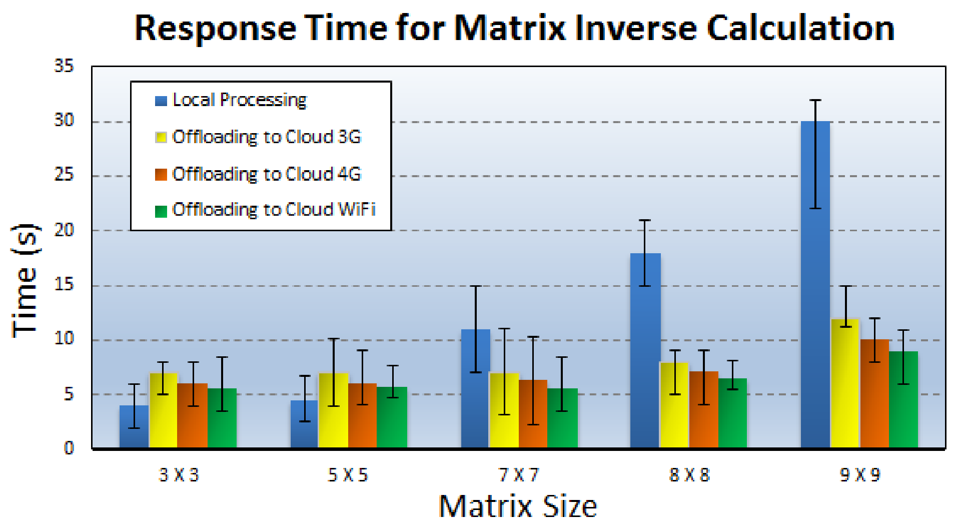


Figure 4.3: Response Time for Matrix Inverse Calculation

of matrix increases in local processing mode, largely because there is an increase in CPU's energy consumption as number of floating point operations increase. This application calculates matrix inverse using Adjoint method. As we can see in the Figure 4.2 and 4.3, offloading the processing for matrix calculation on Cloud saves energy as the matrix size increases, but the local processing is suitable for small matrix operations (i.e. 3X3 and 5X5), as it saves both energy and time.

Offloading mode saves maximum energy when used with Wi-Fi. 3G and 4G cost almost the same energy wise but 4G performs better on response time of the operation.

4.2.2 Internet Browsers

Cloud based Internet Browsers were introduced in order to overcome the processing and energy limitations of mobile devices. Already there are a number of cloud-based mobile web browsers that are available in the industry e.g. Amazon Silk [11], Opera Mini [12], Chrome beta [13] etc.

Let us understand more about these browsers first. Cloud-based Web browsers([11], [13], [12], [14]) use a split architecture where processing of a Mobile web browser is offloaded to cloud partially, it involves cloud support for most browsing functionalities such as execution of JavaScript , image transcoding and compression, parsing and rendering web pages. Prior research in this area such as [15] shows that Cloud based Internet browsers do not provide clear benefits over Local or device-based browser (e.g. Local Processing) either in energy or download time.

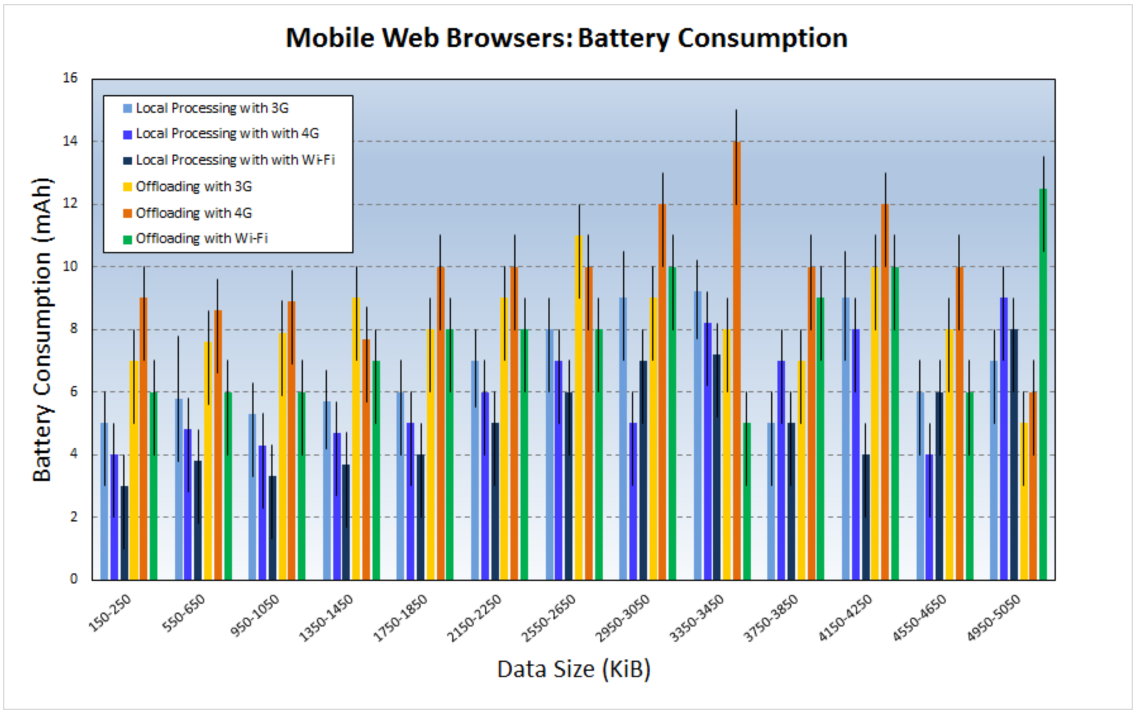


Figure 4.4 Battery Consumption for Web Browsers

Offloading JavaScript to the cloud is not always beneficial, especially when user interactivity is involved [15].

We used one of the commercially available Cloud based mobile browser and a Local browser for our experiments. In Figure 4.4 and 4.5, we have plotted the results obtained by measuring data transfer and response time required by these browsers for following websites: 1. www.yahoo.com, 2. www.wikipedia.org, 3. www.amazon.com, 4. www.google.com, 5. www.facebook.com.

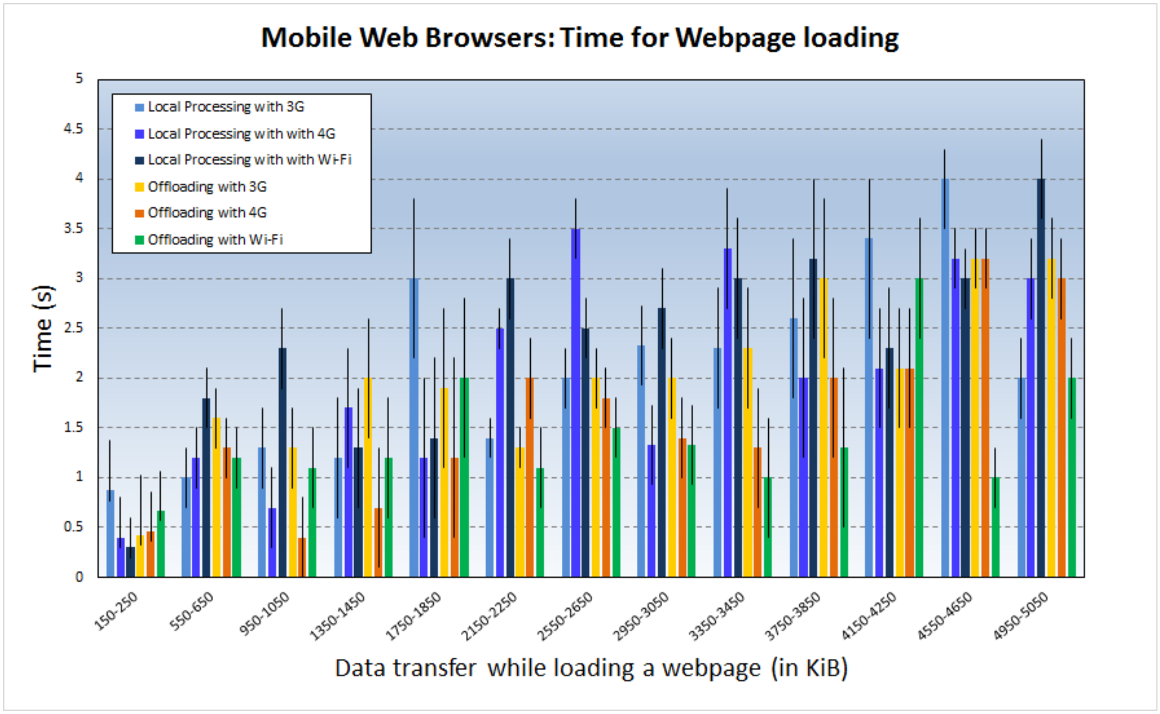


Figure 4.5 Response Time for page loading in Web Browsers

Results obtained are for a data range starting as low as 150 Kib to a session involving 5 MBs of data transfer to load the webpages. We can observe here that Cloud based web browsers are faster but expensive in terms of energy consumption. For small data transfers it is always suitable to use Local web browser to save both time and battery consumption. For a normal user overall data transfer during the browsing session does not go beyond 5-6 MBs for single session, which means we always will have small data transfers to the cloud and Local browsers show better results for those cases. If we compare energy consumption, Local processing mode performs better in most cases but not all. Response time results does not declare the offloading mode a winner either.

Interestingly 4G energy consumption is higher than 3G and Wi-Fi in most cases, but not always. Wi-Fi performs better than 3G and 4G most of the times, when in offloading mode.

4.2.3 Zipper

Here the idea is the processing of zipping the files will be done either locally or on the cloud as directed by the Decision engines. The Zipper is an Android app that we used to compress the files locally. For Cloud based file compression, we used online zipping tools such as [16] and [17].

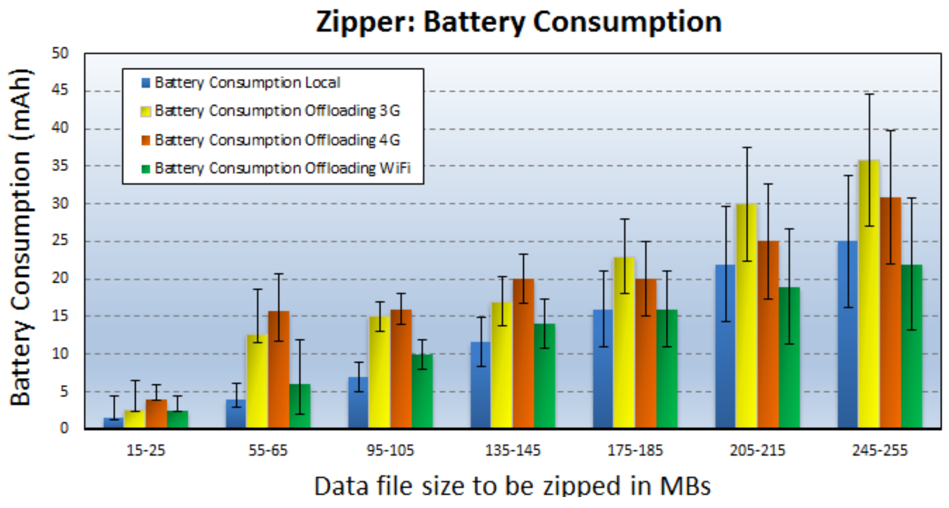


Figure 4.6 Battery Consumption while zipping files

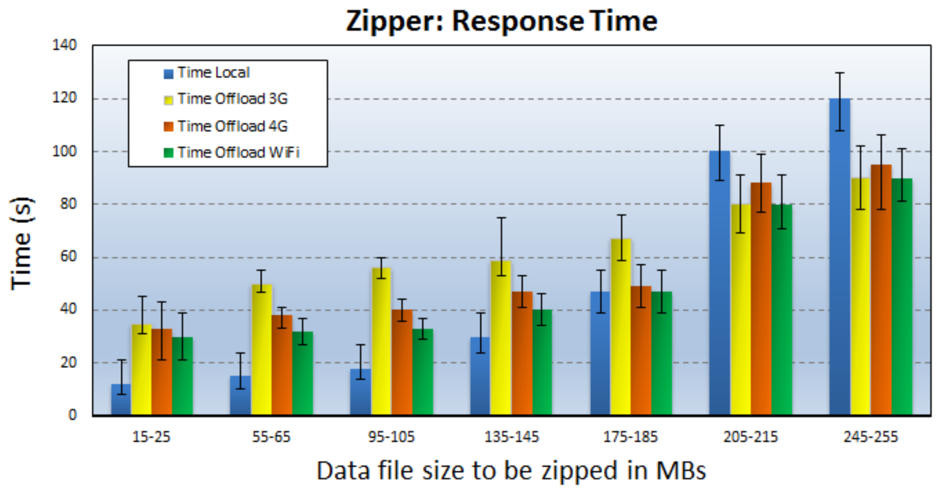


Figure 4.7 Response Time while zipping Files

In Figure 4.6 and 4.7, we compare energy consumption and response time while doing local processing and offloaded processing with varying file sizes. For compressing files, pdf and word document files were used.

4.2.4 Voice Recognition and Translation

There are many smartphone apps which use cloud to do the voice recognition and translation, for instance Google translate. It also has an offline translation mode that does local processing on the device with small a Neural Network. In Figure 4.7, we can see the energy consumption of a Voice Recognition and Translation app on our devices for a range of words. These experimentations are carried on our handsets using 3G, 4G and Wi-Fi networks for recognizing and translating 20-140 words from English to Marathi translations.

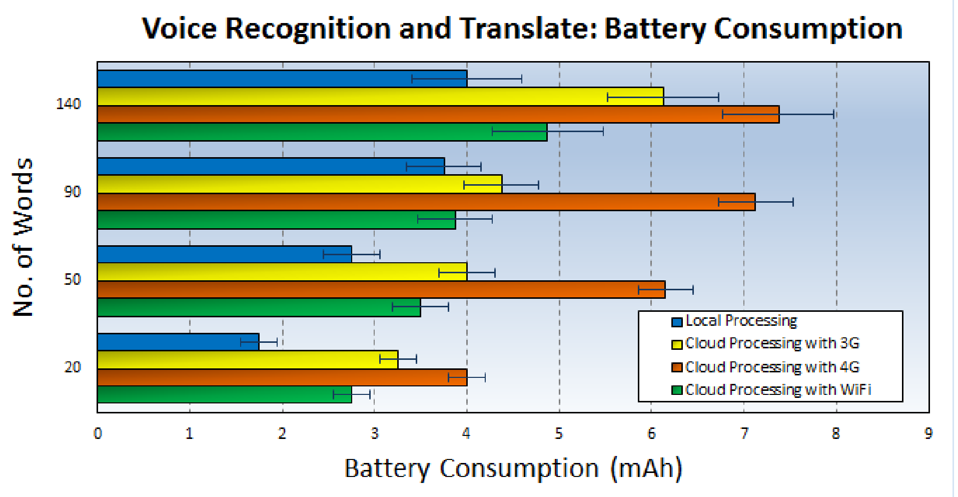


Figure 4.8 Battery Consumption while Voice Recognition and Translation App

4.2.5 Torrents

In this strategy, the cloud servers are used as a BitTorrent client to download torrent pieces on behalf of a mobile handheld device. While the cloud server downloading the torrent pieces, the mobile handheld device switch to sleep mode until the cloud finishes the torrent processes and upload the torrent file in one shot to the handheld device. This strategy saves Energy of smartphones because downloading torrent pieces from torrent peers consumes more energy than downloading a one burst of pieces from the cloud. Similar strategy is proposed by Kelenyi et al. in [18].

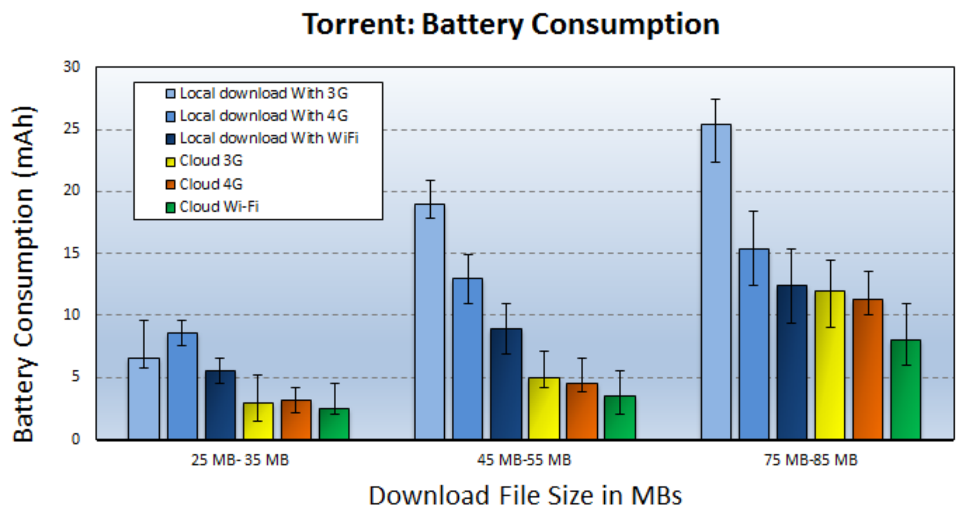


Figure 4.9 Battery Consumption while Torrent downloading

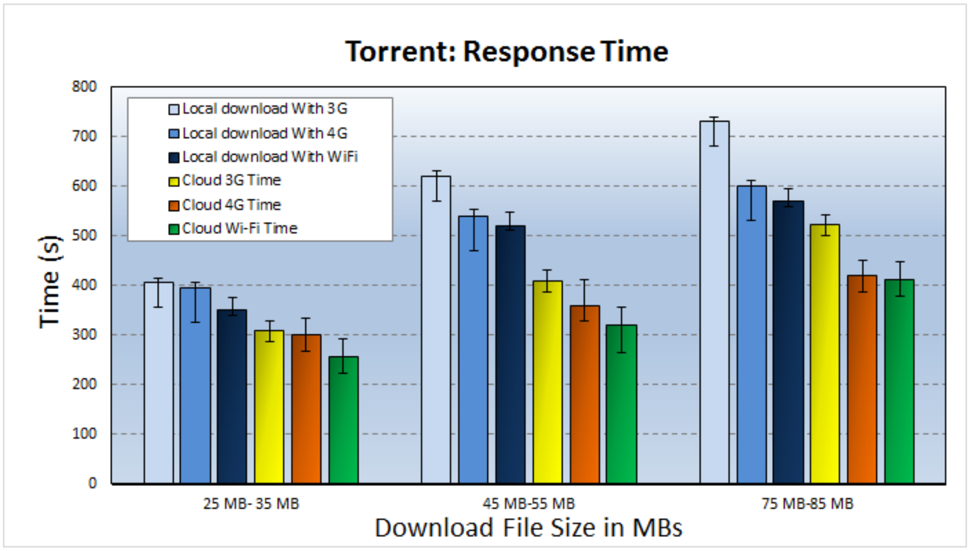


Figure 4.10: Response Time while Torrent downloading

4.3 Summary of Findings

In the previous publications such as [2], the offloading is shown to be useful when an application is compute intensive at the same time not data intensive. Although this is true, it is not the complete picture. In our experiments after studying various real time applications we have found that the cloud computing is more beneficial for the applications which may not be compute intensive but data intensive, for example Torrent download application.

If the application uses the display all the time, then chances are that offloading will not be beneficial. This is because the display will consume much higher energy than all the other components such as CPU, sensors put together, and therefore the CPU power saved by offloading (if any) will be fractional as compared to the display.Video processing is a good example for this case.

Offloading does better in terms of both energy and response time for applications which do not use display intensively and are at the same time compute intensive (for example Matrix app). Internet browsers are neither highly compute or data intensive, in such case offloading mode doesn't perform well.

To make offloading more practical it is important to reduce the energy spent in the communication between mobiles and the cloud. In order to do so it is crucial to manage the process of choosing best possible network. Therefore it is important to compare energy consumption in mobiles with varying networks like 4G vs Wi-Fi or even 4G vs. 3G. One may assume that because 4G is fastest and we can always rely on it for offloading; however our results clearly indicate that it is not always the case. Surely, it depends on 4G band, carrier and device.

If you have a perfect 3G coverage as opposed to poor 4G than 3G would perform far better than 4G and vice versa. If you live between cell edges or where a coverage of 3G/4G ends then handovers will kill your battery. 4G generally means faster data rates and as a result user tend to consume more data; this could lead to battery draining much more. Depends on the amount and type of usage, signal strength in the area and the kind of apps that are being used. Some apps require a channel to be established between the base station and the mobile phone at regular intervals, which drains the battery.

Your phone is constantly pining for the network. That means it periodically scans the airwaves around it to determine which tower it should tether itself to. The more networks there are to choose from the more scans it must make. 4G phones are fast, but they can also suck a battery dry in a few hours. The radio in the 4G or LTE device is doing a lot more than it ever did in your old 3G handset. The radio is the single biggest source of power drain in any device apart from the LED screen, but unlike the display, the radio is always on. And 4G is particularly hungry. Most of the 4G devices sold today use a technology called MIMO, which doesn't just send or receive a single signal, but rather has multiple parallel transmissions. Each phone has two antennas, each of which requires its own power amplifier.

In the next section we have proposed a middleware framework for offloading decision engine mechanisms which can be used by a smartphone to decide which network carrier will be the best for offloading.

CHAPTER 5

MIDDLEWARE FRAMEWORK FOR SMART OFFLOADING

In this chapter, we present a machine learning based middleware framework to enhance the overall offloading process. Prior works such as [8], [9] have proposed an offloading decision engine that will consider the parameters on the device and on cloud to make a correct offloading decision. An offloading decision engine requires a consistent network performance for offloading. However, such consistency is difficult to achieve because of frequent mobile user movements and unstable network quality, which varies, depending upon the Location of the device, load on the network. The power consumed by the network radio interface is known to contribute a considerable fraction of the total device power.

With recent advent of 4G networks, there has been increased interest in the offloading domain, but our results show that 4G is less power efficient compared to Wi-Fi, and even less power efficient than 3G which is also confirmed in some of the prior works [19]. 4G phones are supposed to be faster, but that is not always the case.

5.1 Need to choose right network while offloading because of Network Inconsistency

With the advances in networking technology, we have 4G available to us, but it is seen that 4G consumes more energy than 3G and Wi-Fi.

In general, anything involving transferring large amounts of data gets a big boost from 4G. If the device is in an area that does not have 4G coverage, there is no advantage to a 4G phone, and if we do not disable 4G, the radio's search for a non-existent signal will drain the battery quickly.

Level of connection on 4G at your current location will greatly affect your battery life. Meaning, if you have a weak signal, your device will be using more power to get data sent and received to and from the network, which will eat your battery up. A strong 4G signal will of course use less battery; the biggest problem is the constant switching from 4G to 3G and back again. On the other hand, different network carriers have varying performances even if they are of the same category as 3G or 4G. Not only that even with changing time the performances of the network varies because of changing load on network traffic. We can call all these problems as 'Network Inconsistency'.

To counter such Network Inconsistency to some extent on the device and to optimize the offloading experience we propose a novel offloading technique based on Machine Learning which helps a device choose between the available networks with varying conditions, so that we get consistent results in offloading as far as possible. In the next Section, we have described this system in detail.

5.2 Reinforcement Learning (RL) Decision Engine

In this section, we explored Reinforcement Learning to create a decision engine for the offloading process. Reinforcement learning is learning by interacting with an environment. Reinforcement Learning (RL) differs from standard supervised learning in that correct input/output pairs are never presented. Reinforcement Learning is already described in great detail in the Chapter 3 of this thesis.

Here we are using RL for a Single-step decision problem; RL can be used for Multi-Step decision problems. We have explained how the offloading decision process can benefit from Reinforcement Learning and presented different scenarios where it can be used to improve the overall offloading decision process.

5.2.1 State-Action Value Function

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.

a) Set of Possible State Values

In our Algorithm State Values are discrete values.

- Location = Home, Office, Traveling
- Data Transferred = Data_Small, Data_Medium, Data_Large
- Time = Morning, Afternoon, Evening, Night

The offloading system extracts the above parameters (such as Location, Time) from the contextual information of the Smartphone device.

b) Set of Action Values

Values of Possible Actions are also discrete values.

- Offload using 3G

- Offload using 4G
- Offload using Wi-Fi

c) Representing the Q Table with Penalty values

Say we are in state S_t at time t. Upon taking action at from that state, we observe the one-step reinforcement p. After this we choose another action a_t+1 in the same state, this continues until we have explored all the Actions. The cost or Penalty of an action taken from a state is the reinforcement for that action from that state. Use the returned Penalty value to choose best Action, where we want to minimize the Penalty. When there is a change in User's Location, for instance User moves from Home to Office, We continue the same steps mentioned above.

	Offload Using 3G	Offload using 4G	Offload using Wi-Fi
P_b	P_{b3G}	P_{b4G}	P_{bWIFI}
P_t	P_{t3G}	P_{t4G}	P_{tWIFI}

$$p = P_{b3G} * x + P_{t3G} * y$$

P_b	- Penalty for Battery units consumed
P_t	- Penalty for time required to offload
P_{b3G}	- Penalty for Battery units consumed while offloading with 3G
P_{t3G}	- Penalty for time to do the offloading operation with 3G
P_{b4G}	- Penalty for Battery units consumed while offloading with 4G
P_{t4G}	- Penalty for time to do the offloading operation with 4G
P_{bWIFI}	- Penalty for Battery units consumed while offloading with Wi-Fi
P_{tWIFI}	- Penalty for time to do the offloading operation with Wi-Fi

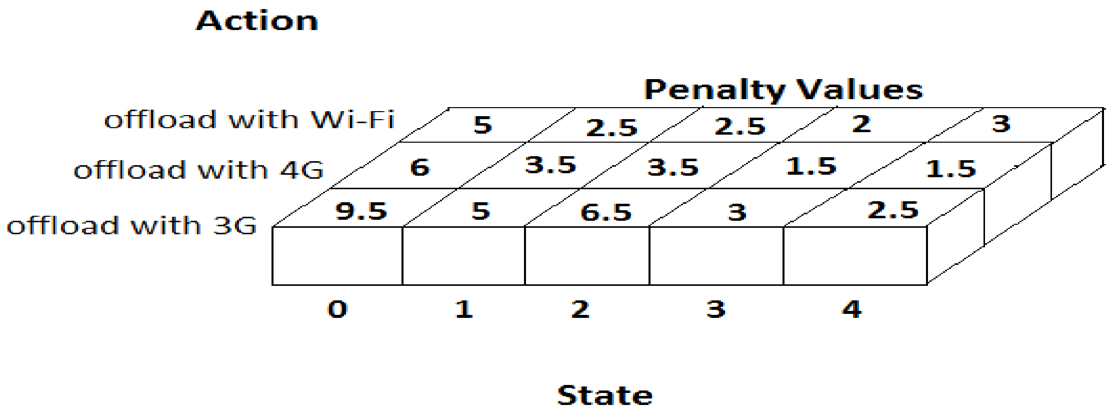


Figure 5.1 Representing the Q Table with Penalty values

RL Algorithm:

1. Detect change in the Device's contextual information (Parameters such as Location (Home, Office, etc.) and Time-Period (Morning, Afternoon, Evening and Night))

2. Activate 3G radio interface of the device.

3. Download a file (data_size = data_small) from the Cloud. Measure the Battery and time consumed for the operation.

4. Upload same file on the cloud. Measure the Battery and time consumed for the operation.

5. Calculate the penalty p with the help of equation in Figure 5.1

6. Form a Key-Value pair as follows:

{Location-TimePeriod-data_size: penalty} where

Key = Location-TimePeriod-data size

Value = Calculated penalty p

7. Update the Q-table with the calculated penalty values as shown in Figure 5.2.

8. Repeat steps 2-7 above for (data_size = data_medium and data_large)

9. Repeat steps 2-8 above for 4G and Wi-Fi connection if available.

The application with offloading mechanism will refer to the updated Q-table to make a right decision of choosing the network for offloading the required computing and data on the cloud. The application will look for the minimum penalty values available in the Q-table. The values of constants are x = 0.5 and y = 0.5 for both optimized battery and performance time, if user prefers better battery performance than elapsed processing time of the application then we change the values to x = 0.9 and y = 0.1. For example when the user is traveling he might prefer to save his battery power than worrying about the processing time; whereas when the battery charge isn't a sudden problem for the user then he might choose for optimized performance time, in that case we need to change the constant value to x = 0.1 and y = 0.9. The Network Inconsistency is also taken care in our Algorithm as we have included the Location parameters in the state values to train our Q - function.

5.3 RL with Neural Network

In this section we have used Neural Network to test our Reward based Machine Learning mechanism in a simulated environment. We developed a Neural Network code in Python and have demonstrated results. In this algorithm we have simulated 10 different locations a user goes through (Locations 0-9) as shown in the x-axis of topmost Figure in 5.3; on Y-axis we have shown the Reinforcements received by the offloading system, for '-1' the system decides to do local processing because the conditions are not suitable, for '0' we offload the processing on Local servers, and for '1' we offload on remote cloud servers; On Z axis we have shown The

Reinforcements of Response Time of the app processing. As we can see these locations have certain attributes assigned to them with random values. For instance as the user goes from location 0 to 9, the network bandwidth available to him increases. We can specify for how many iterations we want to train our Q - function and also the hidden layers of the Neural Network is customizable. After sufficient training the 3-D Contour of QModel looks as shown in the middle-right portion of the Figure 5.3, below that is just the top view of the contour. On the middle-left Figure are the different trials of the action taken by the offloading system while going from Location 0-9.

- -1 for Local Processing
- 0 for Offloading on Local Servers
- 1 for Offloading on Remote Servers

In the Figure 5.3, we can see the surface plot of our trained Q function. For these set of results we have customized my Neural Network with no. of hidden layers as (nh) = 5, run for trials (nTrials) = 100 and Steps per trial (nStepsPerTrial) = 10. In the first plot on x - axis we have different locations where the devices is in and location point varies from 0 - 10; on y - axis we have plotted

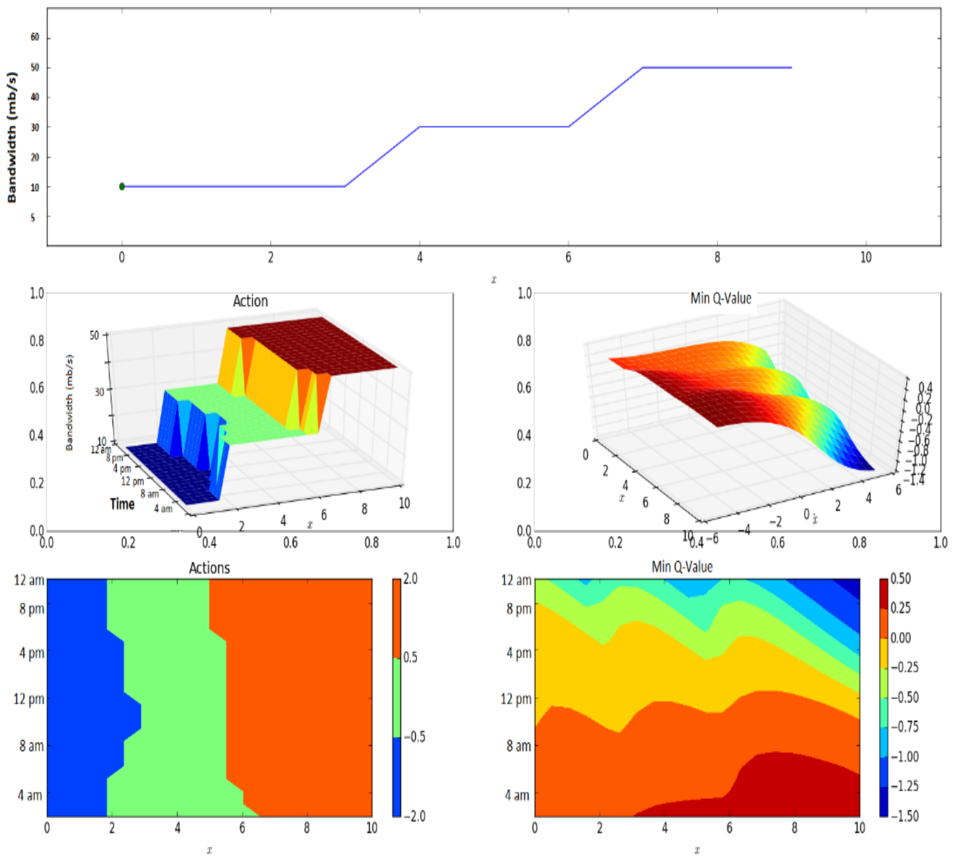


Figure 5.2 Reinforcement Learning (RL) with Neural Network (NN)

the actions recommended by the Q function depending upon the best scenario to save the battery power. Therefore, we can see that for location 0, 2 and 3 Local processing is favored by our Q function. When the user moves to different locations between 4 - 6 the Q functions choose to offload the processing on Local servers whereas for locations 6 - 10 we should offload on remote servers. This decision is based on various parameters values present in that location such as `bandwidth available'.

In the ‘Actions' plot we can see 3-D plot with location and Bandwidth parameters. In the ‘Max Q' plot we can see the maximum values that our Q function has for various locations, the Red part is where we got maximum Reinforcement values.

Reinforcement Learning (RL) is an Unsupervised Learning method, our Algorithm is simplistic in comparison to other works like Smartphone Energizer (SE) [9], which is a supervised learning method. Our results show that we can save up to 20%-30% battery power in the proposed offloading system while we compare it with prior works ([8], [9]).

5.4 Fuzzy Logic Decision Engine

Fuzzy Logic deals with approximate rather than fixed reasoning, and it has capabilities to react to continuous changes of the dependent variables. The decision of offloading processing components to cloud becomes a variable, and controlling this variable can be a complex task due to many real-time constraints of the overall mobile and cloud system parameters.

In [8] the authors have proposed fuzzy decision engine for code offloading, that considers both mobile and cloud variables. We have implemented a similar engine with relevant parameters and with slightly different rules. We have demonstrated the working of the app in Figure 5.4. Java code that was developed for the app is shown in the end. In this section a Fuzzy Logic Decision Engine Implementation is discussed.

5.4.1 Mobile Offloading Logic

At the mobile platform level, the device uses a decision engine based on fuzzy logic, which is utilized to combine n number of variables, which are to be obtained from the overall mobile cloud architecture. Fuzzy Logic Decision Engine works in three steps namely: Fuzzification, Inference and Defuzzification. Let us see these steps in detail:

1) In Fuzzyfication input data is converted into linguistic variables, which are assigned to a specific membership function. 2) A reasoning engine is applied to the variables, which makes an inference based on a set of rules. Finally 3) The output from reasoning engine are mapped to linguistic variable sets again (aka defuzzification).

The Engine considers input parameters from smartphone and cloud, these inputs are divided into intervals. For Example Network Bandwidth is a variable, it is divided into intervals low speed, normal speed and high speed with values (0, 30), (30, 70), (70, 120) in mbps respectively. Other variables are also divided into similar intervals. Fuzzy rules are created based on best experimented resulted variables. For instance a fuzzy rule to offload to remote processing is constructed by combining high speed, normal data, and high CPU instance with an AND operation.

a) Fuzzy sets considered

- Bandwidth = Speed_Low, Speed_Normal, Speed_High
- WiFi = available, not available
- Data transfered = Data_Small, Data_Medium, Data_Big
- CPU instance = CPU_Low, CPU_Normal, CPU_High

Let us see what these parameters define. Bandwidth available to user device can be Low, Normal or High. Data that is used by the application can affect the decision of offloading if the Data is too big the offloading can be expensive. CPU instance required by the application can be Low, Normal and High depending upon the computational requirements of a particular application. If the WiFi is available to the user then it makes more sense to offload on the local servers rather than the remote servers. So here, we are assuming there are multiple locations available with the user where he can offload his application processing and data. After defining these parameters, we have assigned grade of truth values to each of the set considered which fuzzy logic uses to classify the outputs.

b) Rules Considered

- Remote Processing = Speed High AND Data Small AND CPU Normal
- Local Processing = Speed Low AND Data Small AND CPU High
- Remote Processing = Speed Normal AND Data Small AND CPU High
- Local Processing = Speed High AND Data Small AND CPU Low
- Local Processing = Speed Low AND Data Medium AND CPU Normal
- Offload on Local Servers = Remote Processing AND WiFi ON
- Offload on Remote Servers = Remote Processing AND WiFi OFF

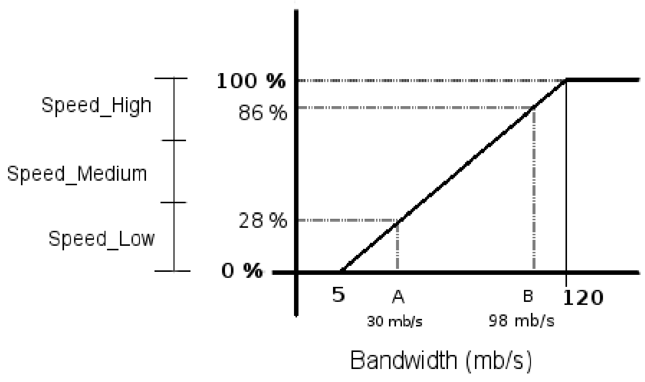
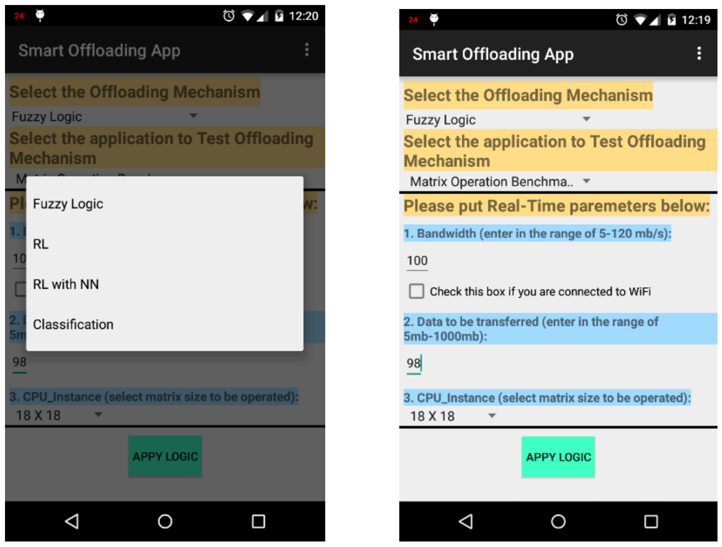


Figure 5.3 Classifying with Fuzzy Logic

A Fuzzy Logic system infers a decision expressed as the degree of truth to a specific criteria. The Grade of truth is the percentage value which helps us classify variables into a specific group. Fuzzy logic engine is fed by information extracted from the code offloading traces. The grade of truth is computed by applying center of mass formula to the decision.



Select the Offloading Mechanism
Fuzzy Logic
Select the application to Test Offloading Mechanism
Matrix Operation Benchma..
Please put Real-Time paremeters below:
1. Bandwidth (enter in the range of 5-120 mb/s):
100
Check this box if you are connected to WiFi
2. Data to be transferred (enter in the range of 5mb-1000mb):
98
3. CPU_Instance (select matrix size to be operated):
18 X 18

Figure 5.4 Offloading Decision Engine in Android

5.5 Comparing our Algorithms with previous works

In this section, we have done a comparison between our strtaegies and other prior works. We have chosen a compute intensive app Matrix Inverse Calculator and a Data intensive app such as Torrent to compare the results as shown in Figure 5.5 and 5.6. We have done our analysis for the Torrent download of a 25 MB file and for 9X9 Matrix Inverse calculator instances.

Our results indicate that the offloading is beneficial for Data Intensive applications such as Torrents. Moreover, it is seen that local processing proves to be beneficial in the Matrix inverse calculator app; whereas the learning strategies like Reinforcement Learning after a suitable initial training time, can improve the offloading performance to some extent in Compute intensive applications and to a larger extent in data intensive applications.

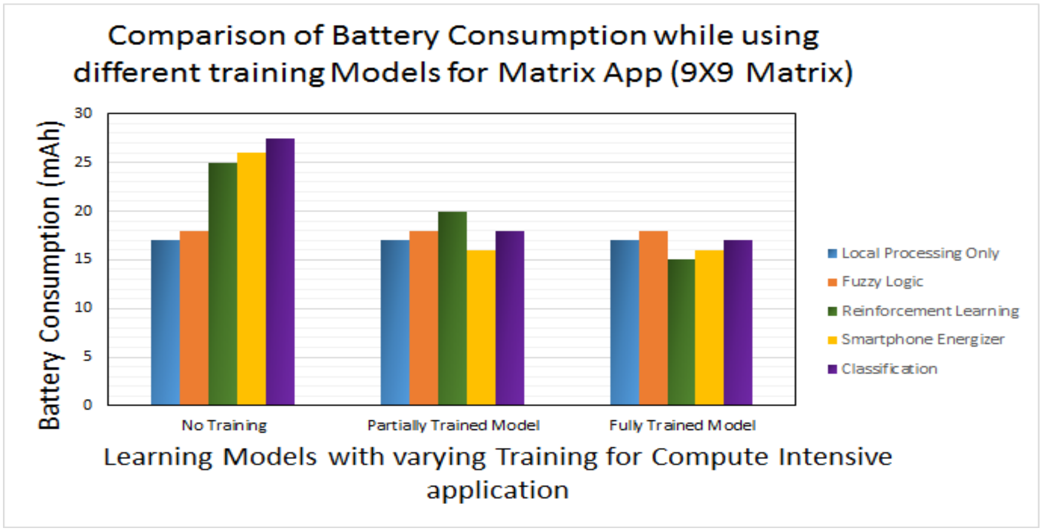


Figure 5.5 Energy Consumption by Different Models for Matrix Operation

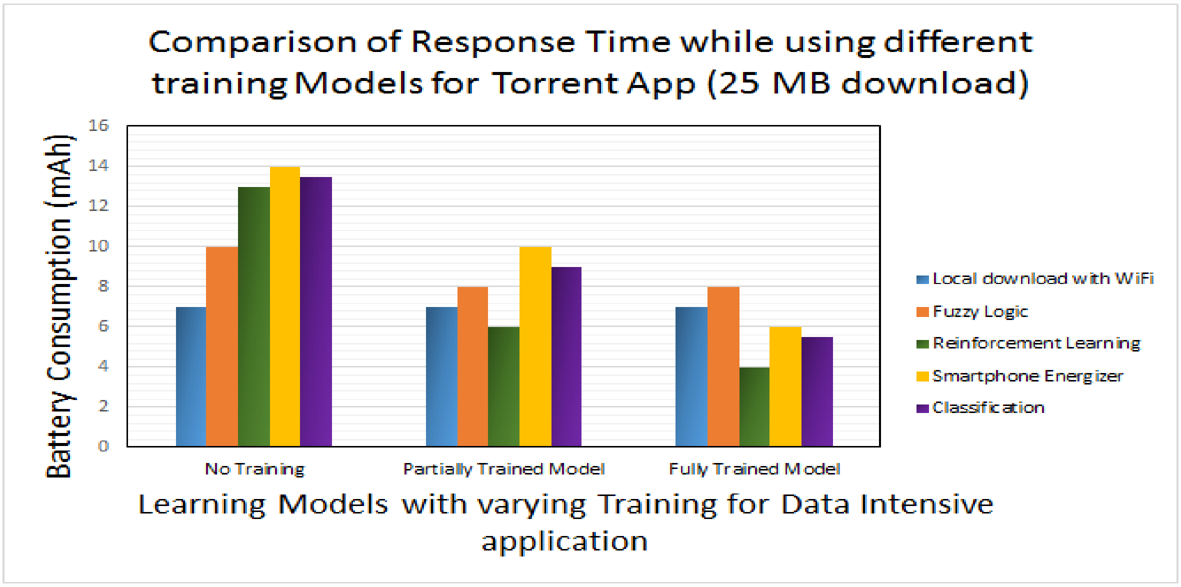


Figure 5.6 Energy Consumption by Different Models for Torrent file download App

CHAPTER 6

SUMMARY AND FUTURE WORK

In the previous research works we have studied that the offloading is useful when an application is compute intensive at the same time not data intensive because then the data need to be transferred will cost more energy to the device. In our experiments we found that cloud computing is more beneficial for the applications which may not be compute intensive but data intensive, for example Torrent downloads app. On the other hand offloading can be beneficial for the compute intensive and data intensive applications such as Face Recognition, Health monitoring applications, etc.. Image processing applications like face recognition require large data sets to train the learning models, which costs energy. Therefore, such applications are very likely to benefit from the offloading process, when the data is already present in the cloud.

offloading Mobile computation on cloud is being widely considered for saving energy and increasing responsiveness of mobile devices. While we do believe that Cloud applications have great advantages, but when it comes to energy savings, our results indicate that offloading does not provide clear benefits over local processing always and therefore use of offloading should be restricted to a small set of applications. We support this claim with the help of experimenting applications such as Cloud-based Web Browser, Voice Recognition and Translation.

In this thesis, we presented Reinforcement Learning based intelligent, mobile network aware middleware framework 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.

6.2 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. This section discusses some of the future work that will enable computation offloading to reach its full potential.

Plans for future work include implementation of the energy-saving machine learning techniques on a real mobile device. The results in Chapter 5 were obtained using a Python implementation of the machine learning algorithms on a 2.6 GHz Intel® Core i5™ processor. To implement the machine learning algorithms in real-time smartphone application is a challenge.

In this thesis we have used only one network carrier to obtain the offloading results, it will be interesting to see the comparison between multiple network carriers. Even in 4G there are different types such as HSPA+ and LTE, in this thesis we have used AT n 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 on 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] MobiThinking, “Global mobile statistics 2011”, http://mobithinking.com, April 2011.

[2] K. Kumar and Y.-H. Lu, “Cloud computing for mobile users: Can offloading computation save energy?”, *Computer*, vol. 43, no. 4, pp. 51-56, 2010.

[3] 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, no. 3, pp. 80-88, 2015.

[4] E. Cuervo, A. Balasubramanian, D. K. Cho, A.Wolman, S. Saroiu, R. Chandra, and P. Bahl, “Maui: making smartphones last longer with code offload,” in *Proceedings of the 8th international conference on Mobile systems, applications*, and services, pp. 49-62, ACM, 2010.

[5] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “Clonecloud: elastic execution between mobile device and cloud,” in *Proceedings of the sixth conference on Computer systems*, pp. 301-314, ACM, 2011.

[6] H. Flores and S. Srirama, “Mobile code offloading: should it be a local decision or global inference?,” in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pp. 539-540, ACM, 2013.

[7] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” in *INFOCOM, 2012 Proceedings IEEE*, pp. 945-953, IEEE,2012.

[8] H. R. Flores Macario and S. Srirama, “Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning,” in *Proceeding of the fourth ACM workshop on Mobile cloud computing and services*, pp. 9-16, ACM, 2013.

[9] A. Khairy, H. H. Ammar, and R. Bahgat, “Smartphone energizer: Extending

smartphone's battery life with smart offloading," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, pp. 329-336, IEEE, 2013.

[10] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*, pp. 59-79, Springer, 2012.

[11] "Amazon silk split browser architecture" https://s3.amazonaws.com/awsdocs/AmazonSilk/latest/silk-dg.pdf.

[12] "Opera mini architecture and JavaScript" http://dev.opera.com/articles/view/opera-mini-and-javascript/.

[13] "Data compression proxy in android chrome beta" https://developers.google.com/chrome/mobile/docs/ data-compression.

[14] X. S. Wang, H. Shen, and D. Wetherall, "Accelerating the mobile web with selective offloading," in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pp. 45-50, ACM, 2013.

[15] A. Sivakumar, V. Gopalakrishnan, S. Lee, S. Rao, S. Sen, and O. Spatscheck, "Cloud is not a silver bullet: A case study of cloud-based mobile browsing," in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, p. 21, ACM, 2014.

[16] "ezyzip: The simple online zip tool." http://www.ezyzip.com/.

[17] "Online-conver.com." http://archive.online-convert.com/convert-to-zip.

[18] I. Kelenyi and J. K. Nurminen, "Cloudtorrent-energy-efficient bittorrent content sharing for mobile devices via cloud services," in *Proceedings of the 7th IEEE on Consumer Communications and Networking Conference* (CCNC), vol. 1, 2010.

[19] 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," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 225-238, ACM, 2012.

[20] Monsoon Solutions Inc., official website, http://www.msoon.com/ LabEquipment/PowerMonitor, 2008.

[21] M. Moller, "Efficient Training of Feed-Forward Neural Networks," Ph.D. dissertation, CS Dept., Aarhus Univ., Arhus, Denmark, 1997.

[22] "The Statistical Portal" www.statistica.com/statistics, Jan 2016.

Appendix 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. Sections A.1 provide 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.

A1. 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);
    }
}
```

A2. 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);
    }
}
```

A3. 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---------------
```

A4. RLwithNeuralNetwork.py

```
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()
```

A5. Cloud Interaction with Python code

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)
```

A6. AWS Cloud Interaction with Shell script

```
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
```

ABBREVATIONS

3D	3-Dimensional
3G/4G	3rd and 4th Generation (of cellular mobile networks)
AP	Access Point
API	Application Programming Interface
ARM	Advanced RISC Machine
AURA	Application and User interaction Aware (framework)
CHBL	Change Blindness
CPU	Central Processing Unit
D(V)FS	Dynamic (Voltage) Frequency Scaling
EDGE	Enhanced Data rates for GSM Evolution
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSM	Global System for Mobile communications
HBLP	High Burst Long Pause

HTC	High Tech Computer corporation
KNN	K-Nearest Neighbor
LBA	Location-Based Application
LCD	Liquid Crystal Display
LDA	Linear Discriminant Analysis
LLR	Linear Logistic Regression
MDP	Markov Decision Process
MEMS	MicroElectroMechanical System
MHz/GHz	MegaHertz/GigaHertz
MVSOM	Missing Values Self-Organizing Map
mW	milliWatts
NN	Neural Network
OLED	Organic Light-Emitting Diode
OS	Operating System
PC	Personal Computer

PCA	Principle Component Analysis
PCB	Printed Circuit Board
PCM	Phase Change Memory
PDA	Personal Digital Assistant
QoS	Quality of Service
RAM	Random Access Memory
RIM	Research In Motion
RISC	Reduced Instruction Set
RSSI	Received Signal Strength Indicator
SCG	Scaled Conjugate Gradient
SDK	Software Development Kit
SMS	Short Message Service
SVM	Support Vector Machine
UI	User Interface
UMTS	Universal Mobile Telecommunications System

VM	Virtual Machine
VRL	Variable Rate Logging
Wi-Fi	Wireless Fidelity