Report
An application oriented study of offloading while choosing between available Network (3G, 4G and WiFi)

Aditya Khune

November 22, 2015

Contents

Abstract

Longer battery life is a key feature demanded by the users of Smart Mobile Devices 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. Due to the advancement in Cloud Computing, the offloading of smartphone applications on cloud has generated a renewed interest among Smartphone Research community.

Offloading or Cyber foraging has been widely considered for saving Energy and increasing responsiveness of mobile devices in the past. In the offloading model, a mobile application is analyzed so that the most computational expensive operations can be identified and offloaded for remote processing. However, 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.

In this work we have presented a Reinforcement Learning(RL) based Offloading System that considers suitable information on the device to make accurate offloading decision. Our method leads to optimized energy consumption and response time. Reinforcement Learning is an approach where an RL agent learns by interacting with its environment and observing the results of these interactions. The reinforcement signal that the RL-agent receives is a numerical reward, which encodes the success of an actions outcome, and the agent seeks to learn to select actions that maximize the accumulated reward over time.

In our strategy we proposes an application and user interaction aware middleware framework that uses Reinforcement Learning(RL) and Fuzzy Logic methods for making Offloading Decisions. We have also used Neural Network to test our Reward based Machine Learning mechanism in a 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. The effectiveness of these techniques is shown by prototyping them for the Android smartphone platform.

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. An average American spends around 2 hours and 40 minutes a day on their smartphone [1]; and unfortunately most of the time these 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. But 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.

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 the availability of infinite resources, and it mainly operates with utility computing model, where consumers pay on the basis of their usage. Vast amount of applications such as social networks, location based services, sensor based health-care apps, gaming apps etc. can benefit from mobile Cloud Computing.

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 onto cloud.

Currently most of the research work in this area is focused on providing the device with a offloading logic based on its local context. In this work we have done an application oriented study of offloading which also involves the results of offloading with variety of available networks (3G, 4G and WiFi). We have proposed a Reinforcement Learning (RL) based system which will help smartphone choose the right network.

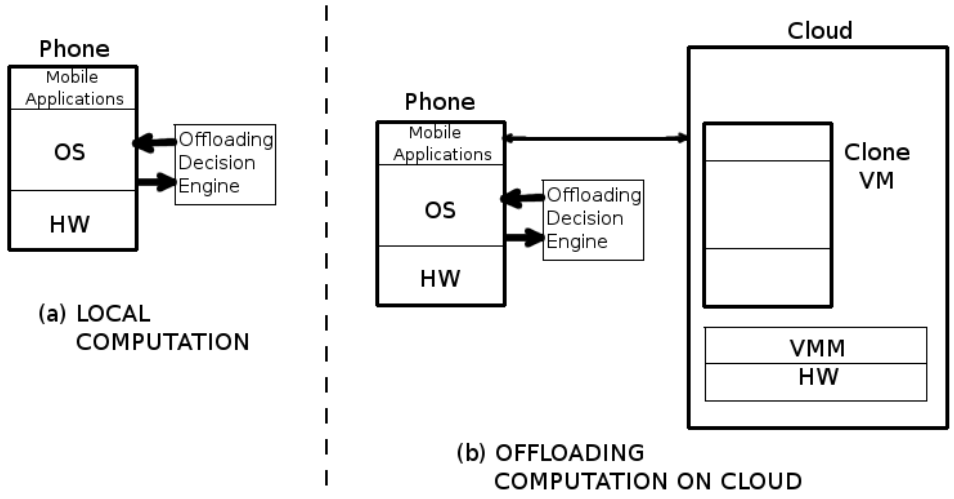

Figure 1.1: Offloading System Model

1.1 Background

The energy saved by computation offloading depends on the wireless bandwidth B, the amount of computation to be performed C, and the amount of data to be transmitted D. Existing studies thus focus on determining whether to offload computation by predicting the relationships among these three factors [2].

Multiple research works have proposed different strategies to empower mobile devices with Intelligent Offloading System.

We have shown a basic offloading architecture in the Fig. 1.1. Offloading relies on remote servers to execute code delegated by a mobile device.

In the architecture that is presented in [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.

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 [4] special static annotations are used by a software developer to select the code that should be offloaded. In [5] authors 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.

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 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 Code 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 a major concern while we want to offload the processing on cloud. Network Inconsistency is studied in detail in Chapter.

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 tried to address some of these challenges by using Reinforcement Learning(RL) Decision Engine for Offloading Process.

1.3 Contributions

- In this work, we present a novel adaptive offloading technique called Reward Based Offloading System 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 an 'Unsupervised learning' method. We have also used Neural Network to test our Reward based Machine Learning mechanism in a Simulated environment.

- Further we have used 'Supervised learning' methods such as Linear Logistic Regression and Linear Discriminant Analysis to create the offloading decision engines.

- We have also compared 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 Engine, and other supervised Learning techniques.

- We have studied the viability of offloading solutions, with the help of benchmark applications to analyze right kind of applications which may benefit from effective Offloading Architecture. During the evaluation, we run these applications under various situations (i.e. different locations, usage, and networks characteristics).

1.4 Outline

The rest of the Report is organized as follows:

- In chapter 2 We have given an overview of Prior Works in Offloading Domain.
- We have conducted extensive experiments in order to do application oriented study in chapter 4. We have presented our results of five selected applications which were tested in different network scenarios.
- In chapter 5 we discuss about the need to choose right network from the available networks in order to be able to save energy consumption and response time on smartphones. we have discussed about the use of Reinforcement Learning(RL) in creating an offloading decision engine for smartphones.

Chapter 2

Prior Works in Offloading Domain

A large amount of work has been done in the area of Smartphone Offloading for mobile devices in recent years. Since 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 smartphones energy bottleneck problem. In this Section, we give an overview of the research in computation offloading and 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 basically 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.

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 is like MAUI since it uses dynamic profiling and optimization solver, but CloneCloud goes a step further as partitioning takes place without the developer intervention; application partitioning is based on static analysis to specify the migration and reintegration points in the application.

In [2] authors have formulated an equation of several parameters to measure whether computation offloading to cloud would save energy or not . These parameters are network bandwidth, cloud processing speed, device processing speed, the number of transferred bytes, and the energy consumption of a smartphone when its in idle, processing and communicating states. In this concept paper, the authors only discussed these various parameters and did not experiment their work in a real offloading framework.

More recently, [7] 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 of Offloading Decision Engines. We will study some of their strategies in later part of this report.

Chapter 3

Overview of Machine Learning Techniques used for Offloading Engine

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 O_t, which typically includes the reward r_t. It then chooses an action a_t 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 trade-off. 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

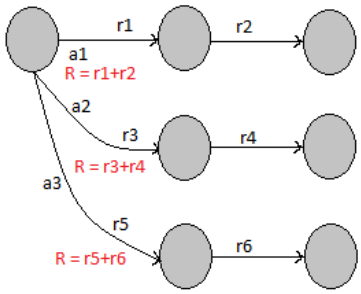

Figure 3.1: Choose from available actions with best Reinforcement History

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 its 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 a_t 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:

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

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

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.

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

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

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 the predicts the expected sum of future reinforcements, because the correct Q function determines the optimal next action.

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

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

This is usually formulated as the least squares objective:

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

3.2 Reinforcement Learning (RL) with Neural Network (NN)

Neural network models in artificial intelligence are usually referred to as artificial neural networks (ANNs); these are essentially simple mathematical models defining a function $f : X \to Y$ or a distribution over X or both X and Y, but sometimes models are also intimately associated with a particular learning algorithm or learning rule. A common use of the phrase "ANN model" is really the definition of a class of such functions (where members of the class are obtained by varying parameters, connection weights, or specifics of the architecture such as the number of neurons or their connectivity).

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.

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.

3.5 Linear Logistic Regression

Similar to LDA, linear logistic regression (LLR) is a technique used to derive a linear model that directly predicts $p(C_k|x_n)$, however it does this by determining linear boundaries that maximize the likelihood of the data from a set of class samples instead of invoking Bayes theorem and generating probabilistic models from priori information.

Chapter 4

Application Oriented Study of Offloading

Offloading has been widely considered for saving Energy and increasing responsiveness of the mobile devices. We have surveyed various applications which are likely to benefit from Offloading as suggested by important publications. Various type of applications mentioned in the important publications are as follows: matrix calculations, natural language translators, speech recognizers, optical character recognizers, image processors, image search, online games, video processing and editing, Web-browsers, navigation, face recognition, augmented reality, etc. These applications consume large mobile battery, memory, and computational resources. Out of those we have listed out 5 applications for experimentations as follows:

1. Matrix Operations
2. Internet Browsers
3. Zipper
4. Voice Recognition and Translation
5. Torrents

We have done energy analysis and response time analysis of all the above smartphone applications, we have compared the results obtained with the help of available network 3G, 4G and WiFi. In the end of this report I have listed out my findings based on the results obtained with all the experimentations. 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.

Smartphone handsets used:

- Samsung S3
- LG G3

Network:

- AT & T's 3G, 4G (HSPA+) Network
- Comcast's WiFi Network

Other Tools:

- Monsoon Power Measurement Tool
- Android Device Bridge (ADB)
- Amazon Web Services (AWS)
- AWS Command Line Interface (CLI)

Experimental Setup and Procedure for plotting the plots:

We have run each experiment 10 times on each handsets mentioned above, and then averaged out the readings obtained. The lower and higher limit of error bars used in our plots is what we obtained by averaging out the readings on the handsets, and what we are showing in the plots is one sample of the readings obtained. Although there isn's a particular standard used in our plots but generally the lower bar energy wise is for LG G3 and higher side of the error bars is for Samsung S3, and vice a versa for Response time plots. All the experiments are done using 3G, 4G and WiFi networks separately in order to understand the effect of choosing the right network while offloading the tasks and data onto cloud.

4.1 Matrix Operations

We have chosen an android app which does matrix operation because most of smartphone applications which include image processing need a processing of large matrices.

This application calculates values of an Inverse Matrix. Figure 4.1 we can see the battery consumption of smartphone increases manyfolds as the size of Matrix increases largely because there 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 4.1 Offloading the processing for matrix calculation on Cloud saves energy as the matrix size increases, but for small matrix operations (i.e. 3X3 and 4X4) the local processing is suitable as it saves both energy and time.

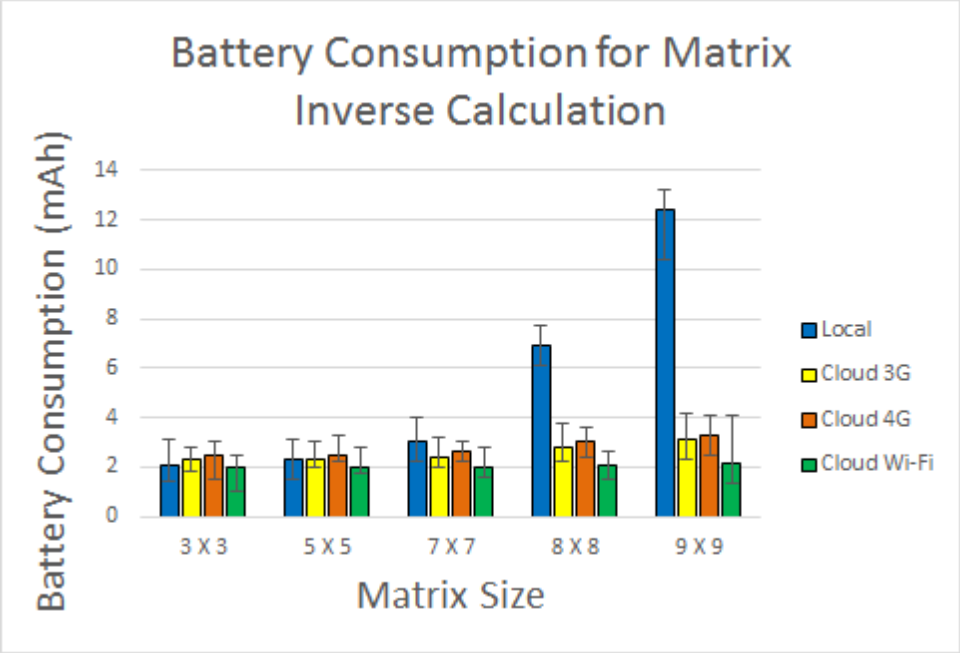

Figure 4.1: Battery Consumption for Matrix Inverse Calculation

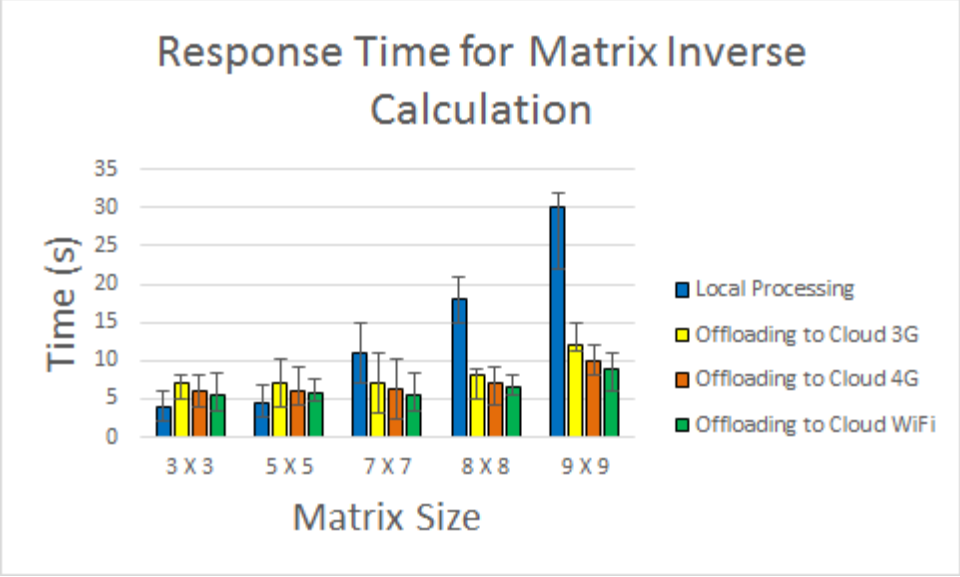

Figure 4.2: Response Time for Matrix Inverse Calculation

4.2 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 (JS), image transcoding and compression, parsing and rendering web pages. Prior research in this area such as [15] shows that CB does not provide clear benefits over Local or device-based browser (e.g. Local Processing) either in energy or download time. Offloading JS to the cloud is not always beneficial, especially when user interactivity is involved [15].

We have chosen one of the commercially available Cloud based mobile browser(puffin) and also a Local browser(Firefox) for our experiments. In Figure 4.3 and 4.4 we have plotted the smartphone readings that we have 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.

We have obtained our readings for a data range starting as low as 150 Kib to a session involving 5 MBs of data transfer to load the webpages. We have observed 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 and that's why Cloud based web-browsers aren't very popular.

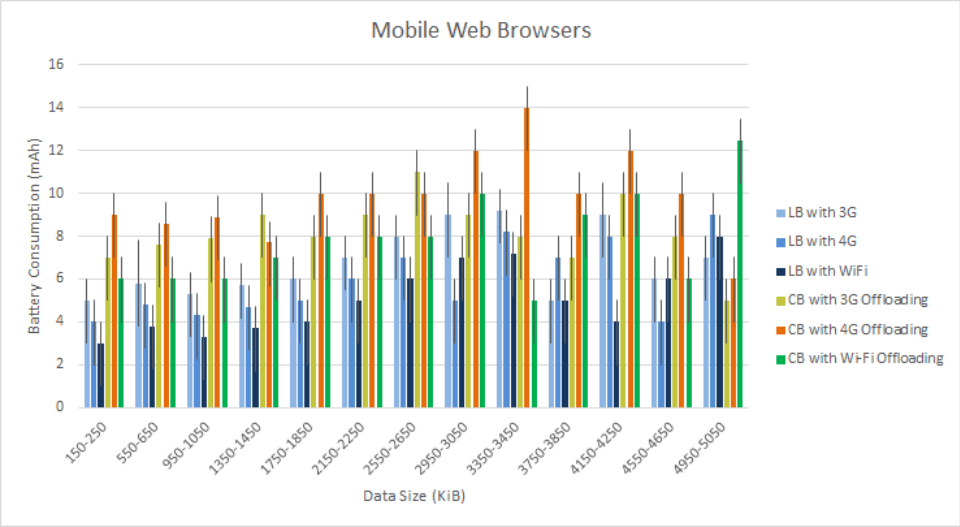

Figure 4.3: Battery Consumption for Web Browsers

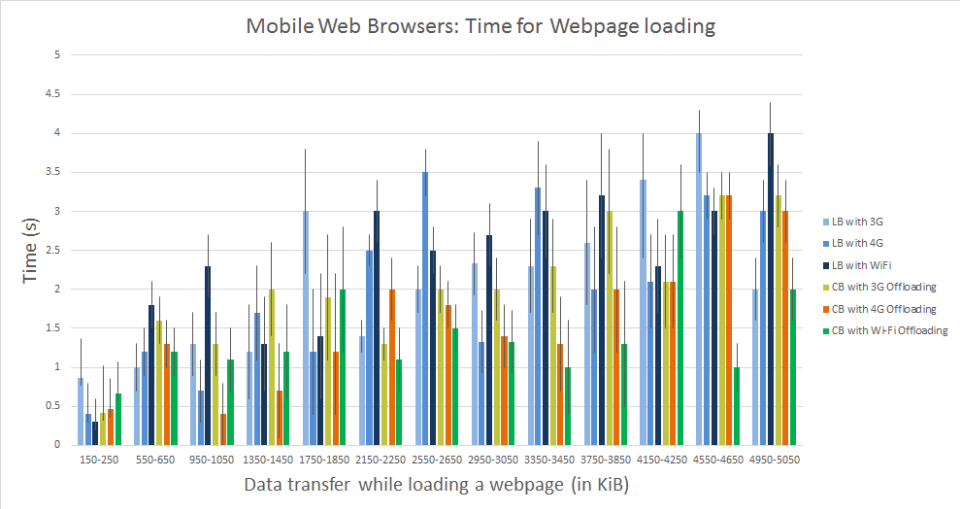

Figure 4.4: Response Time for page loading in Web Browsers

4.3 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 comprssion we have used online zipping tools such as [16] and [17].

In Figure 4.5 and Figure 4.6 we have given a comparison of energy consumption and Response Time while doing Local Processing and Offloaded Processing with varying file sizes. For compressing files we have used pdf and word documents and also MP3 music files in equal size distribution.

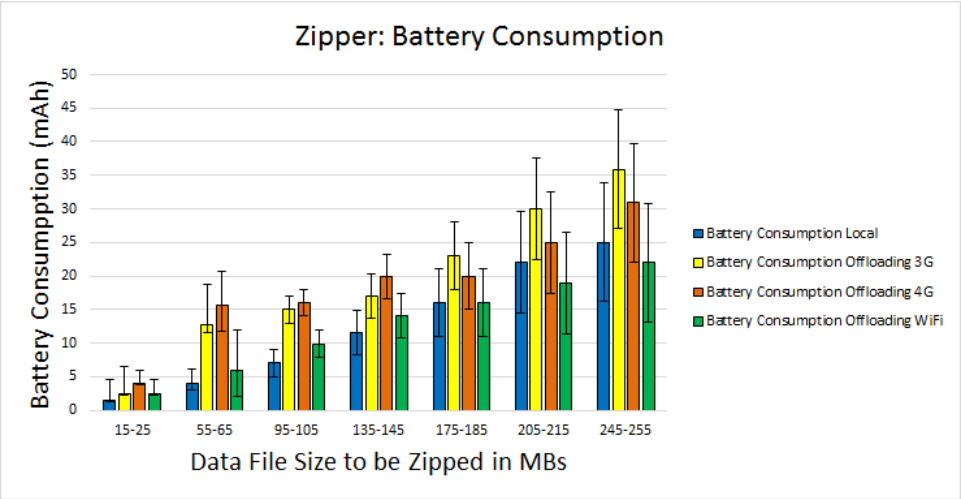

Figure 4.5: Battery Consumption by while file zipping

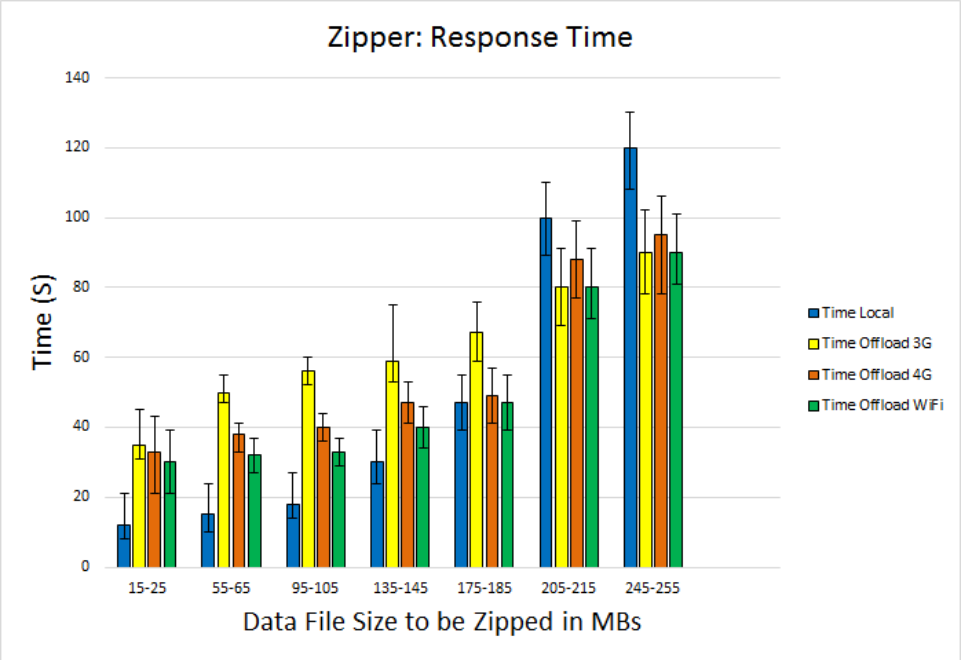

Figure 4.6: Response Time while file zipping

4.4 Voice Recognition and Translation

Google translate is one of the app which uses cloud to do the voice recognition and translation. It also has an offline translation mode which does local processing on the device with small a Neural Network.

In figure 4.7 we can see the energy consumption of this app on our devices for a range of words. We have done our experimentations on our handsets using 3G, 4G and WiFi networks for recognizing and translating 20-140 words from English to Marathi translations.

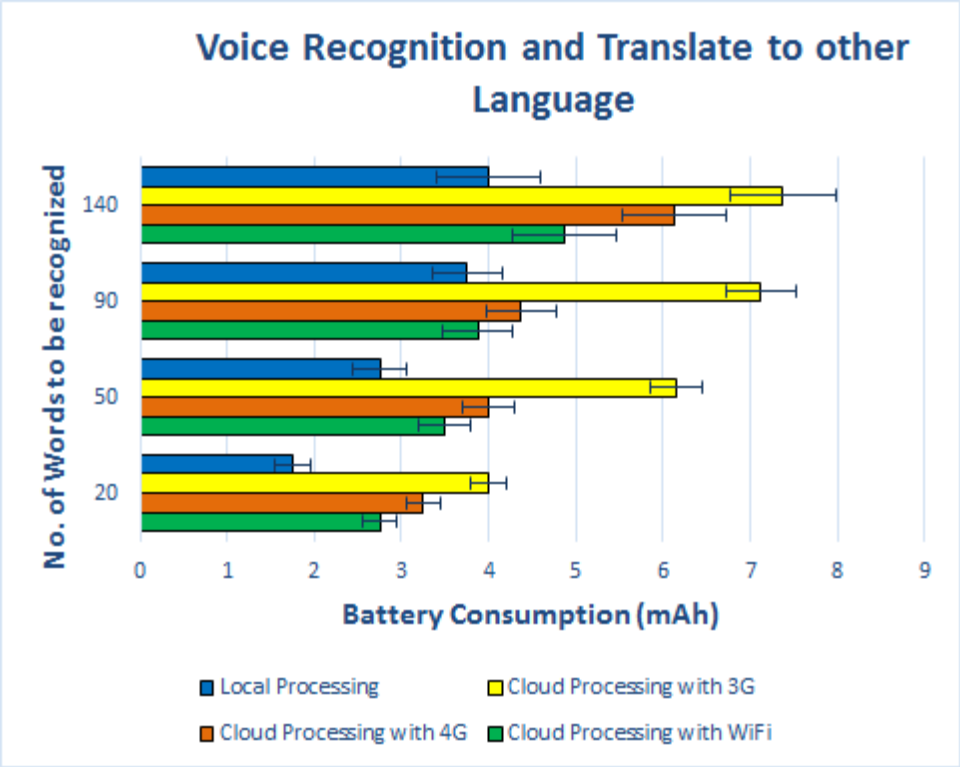

Figure 4.7: Battery Consumption while Voice Recognition and Translation App

4.5 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]

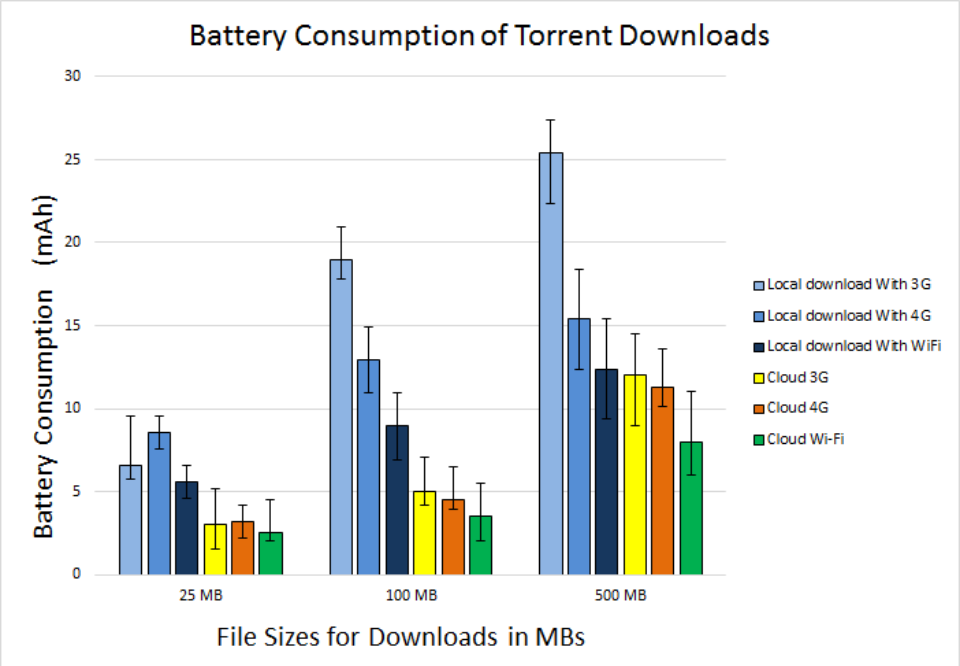

Figure 4.8: Battery Consumption for torrent downloading

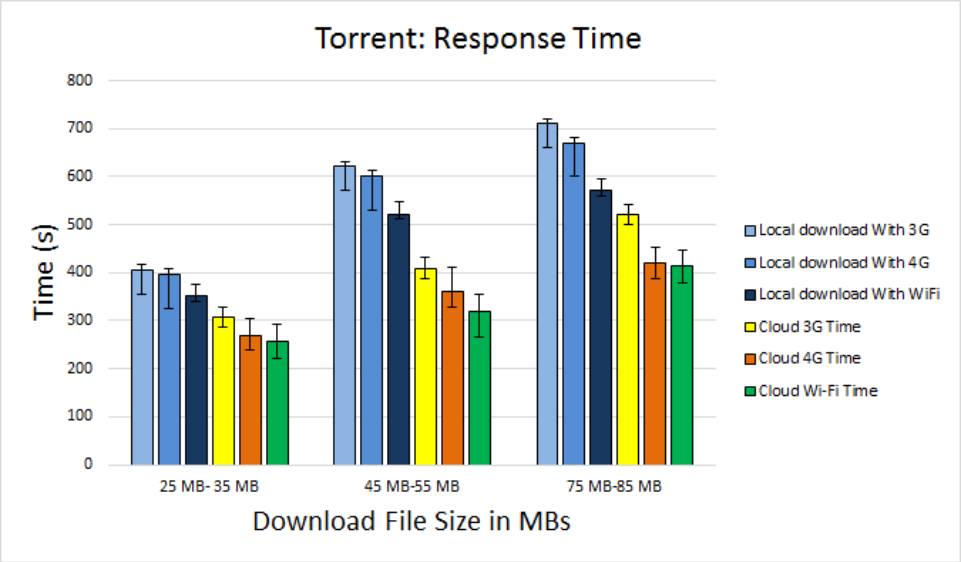

Figure 4.9: Response Time while torrent downloading

4.6 Findings:

Chapter 5

Smart Offloading Decision Engines to Choose between Availble networks and Counter Network Inconsistency

Many prior works in Offloading have proposed an offloading decision engine which 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 LTE networks, there has been increased interest in the offloading domain, but our results show that 4G is less power efficient compared to WiFi, and even less power efficient than 3G which is also mentioned in some of the prior works [19]. 4G phones are supposed to be even faster, but that's not always the case.

5.1 Need to choose right network while Offloading

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

In general, anything involving transferring large amounts of data gets a big boost from 4G. If you live in an area that doesn't have 4G coverage, there's no advantage to a 4G phone. In fact, our experiments show that there are serious battery life problems if you buy an LTE phone and don't disable 4G LTE, as the radio's search for a non-existent signal will drain your battery quickly.

Level of connection on 4g 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.

To counter the Network Inconsistency on the device and to optimize the offloading experience we have proposed a novel offloading technique based on Machine Learning which helps a device choose between the available networks with varying conditions. In the next Section we have described this system in detail.

5.2 Reinforcement Learning(RL) Decision Engine

In this section we have 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.

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

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.

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.

Set of Action Values

Values of Possible Actions are also discrete values.

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

Representing the Q Table with Penalty values

Say we are in state S_t at time t. Upon taking action a_t 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: PenaltyEquation

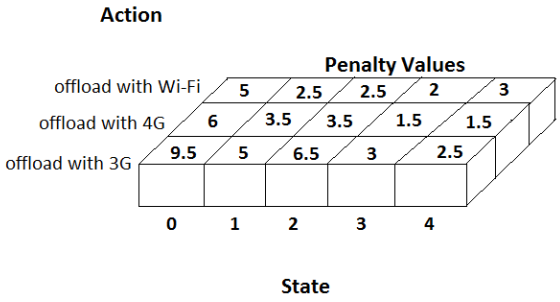

Figure 5.2: Representing the Q Table with Penalty values

Algorithm of our Offloading system

We have developed a partially working prototype of the proposed system as the proof of concept.The prototype implements the basic functionality needed to show the feasibility of the proposed mobile cloud architecture. Below is the 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 ($data\_size = data\_mediumanddata\_large$)
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. I have 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 recieved 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 I 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 $y - axis$ I have plotted the actions recommended by the Q function depending upon the best scenario to save the battery power. So we can see that for location 0, 2 and 3 Local processing is favored by our Q function. When the user moves to

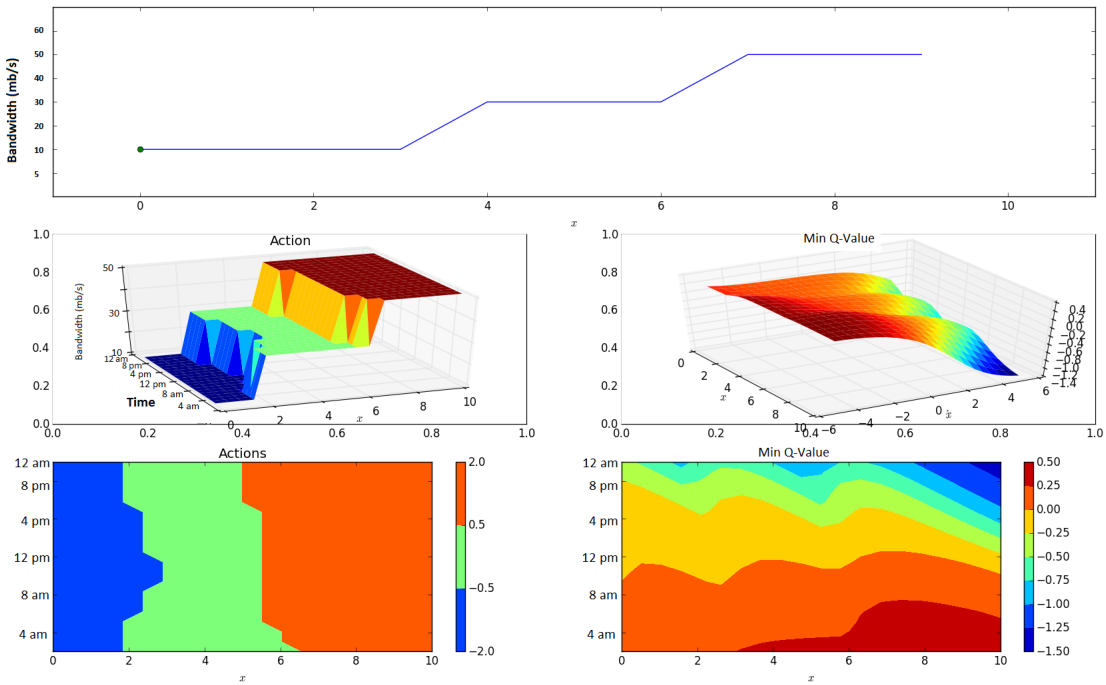

Figure 5.3: Reinforcement Learning (RL) with Neural Network (NN)

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.

Chapter 6

Prior Related Works

6.1 Fuzzy Logic Decision Engine

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.

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).

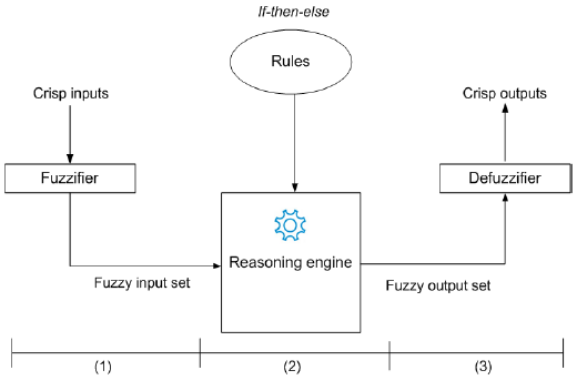

Figure 6.1: Offloading Engine based on Fuzzy Logic

6.2 Problem with this approach

The distribution of different technologies around the world varies significantly. 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. In figure 6.2 we have segmented bandwidth into four different buckets: 0 - 150, 150 - 550, 550 - 2000, and 2000+ kbps. We have mapped them into one of the four Connection Classes Poor, Moderate, Good, and Excellent. Fuzzy Logic is based on simple if-else statements, which is like a hardcoded logic. Because of such variations in the different technologies around the world, it is difficult to rely on the fuzzy logic decision engine presented in [8]. This is because the app developers will have to customize the decision engines depending upon which part of the world the device lies.

6.3 Smartphone Energizer (SE)

Smartphone Energizer [9] is a supervised learning-based technique for energy efficient computation offloading. In this work authors propose a 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

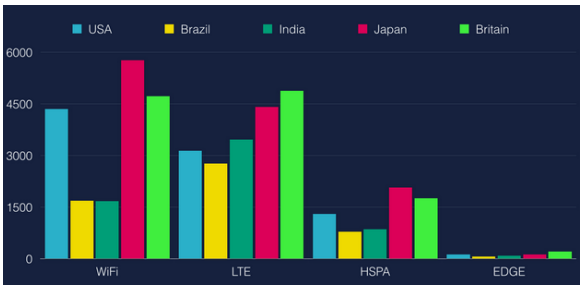

Figure 6.2: Varying Internet Speed across different Countries (Source: Facebook code Community)

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.

6.4 Our work in comparison to the previous works

Reinforcement Learning(RL) is a 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 upto 20%-30% battery power in the proposed Offloading system while we compare it with prior works ([9], [8]).

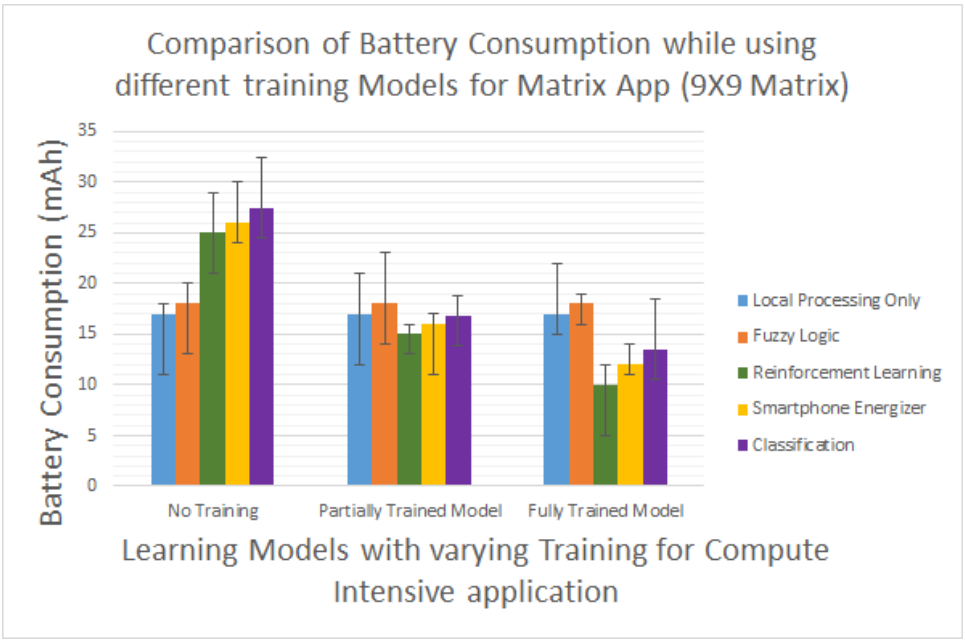

Figure 6.3: Energy Consumption by Different Models with Varying percentage of Training for Matrix Operation

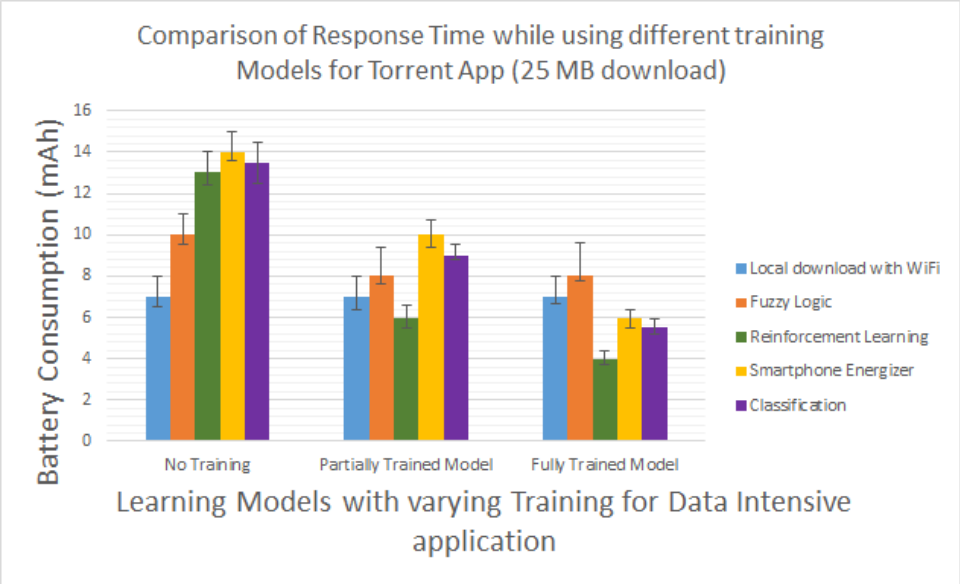

Figure 6.4: Energy Consumption by Different Models with Varying percentage of Training for Torrent file download App

Chapter 7

Conclusion and Future Work

Bibliography

[1] "protectyourbubble.com." http://us.protectyourbubble.com/blog/.

[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. Kelényi 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.