Report Version-1
Reinforcement Learning(RL) based
Smart Offloading System

Aditya Khune

June 20, 2015

Contents

Abstract

Longer battery lifetime is a much desired feature of the mobile device Users today and Low-power design has been an active research topic for many years. 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 partitioned and analyzed so that the most computational expensive operations at code level can be identified and offloaded for remote processing. However, there are many challenges in this domain that are not dealt with effectively yet and the Offloading technique is far from being adopted in the design of current mobile architectures.

Due to the advancement in Cloud Computing, the offloading of smartphone applications on cloud has generated a renewed interest among Smartphone Research community. In this work we have proposed a Reinforcement Learning(RL) based Offloading Engine that considers parameters which are relevant to make accurate offloading decision and which lead 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.

We have presented our results with the help of a Decision Engine app developed on Android Platform. We have analyzed the validity of our algorithm with the help of compute intensive Benchmark applications. We have also studied some of the techniques used by previous works in this area to do a comparative study of our RL based offloading algorithm.

Chapter 1

Introduction

In the last decade we have witnessed phenomenal advancment in mobile technologies. Advances in Mobile devices have changed the way we used our computers. Our personal computers no longer stand on the desks in our homes, but are now live in the pockets of our trousers. The number of applications on our smartphones has exploded over the last several years. Today's smartphones offer variety of complex applications, larger communication bandwidth and more processing power. Unfortunately 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, 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.

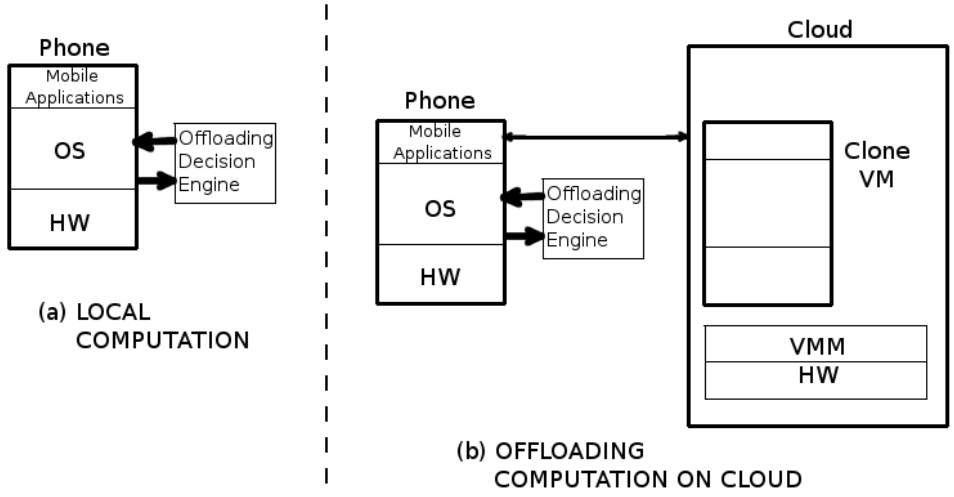


Figure 1.1: Offloading System Model

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 [1].

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 [2] 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 [3] special static annotations are used by a software developer to select the code that should be offloaded. In [4] 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.

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 [5], 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.

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 [2]. 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. [5].

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 [2].

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.

Outline

The rest of the thesis is organized as follows:

- In Chapter 2 We have given an overview of Related Work in offloading domain.

- Chapter 3 gives an overview of Machine Learning Techniques used for Offloading Engine.
- Chapter 4 We have presented our 'Reward based' Offloading Technique and Algorithm
- In Chapter 5 we will see prior works on Smart Offloading Decision Engines, such as Fuzzy Logic Decision Engine, Smartphone Energizer (SE) and Cuckoo.
- Chapter 5 gives some Experiment and Results that are obtained after comparing different techniques
- Chapter 6 Discussion and Future Work

Problem Statement

Offloading of smartphone processing and data on the cloud can be an effective solution to save smartphone resources. Many applications are too computation intensive to perform on a mobile system. If a mobile user wants to use such applications, the computation can be performed in the cloud, whenever the offloading system allows.

The goal of this thesis is to create an accurate Offloading Decision Engine and to optimize mobile device energy consumption via machine learning Offloading management strategies. This thesis discusses multiple strategies.

In contrast to existing works, we overcome the challenges of offloading system in practice by analyzing how a particular smartphone app behaves in real time situations at different locations.

In this work the code of a smartphone app must be located in both the mobile and server as in a remote invocation; a mobile sends to the server, not the intermediate code, but the data to reconstruct that intermediate representation such that it can be executed. We believe that will remove a lot of burden from the app developers. App developing process should be made easy and without much complications so that the developers can focus on the requirement of application and spend less time worrying about annotating their codes and searching for methods in their code which could be offloaded to save the energy.

Chapter 2

Related Work

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 [3] 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 [4] 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 [1] 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, [6] 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.

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

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

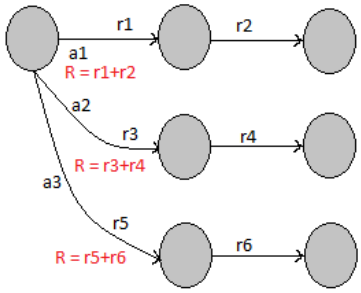


Figure 3.1: Choose from available actions with best Reinforcement History

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 Linear classification using least-squares

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.

Chapter 4

Reward based Offloading

The aim of Machine Learning is to produce intelligent programs or agents through a process of learning and evolving. Reinforcement Learning (RL) is one such approach where an RL agent learns by interacting with its environment and observing the results of these interactions. The idea is commonly known as "cause and effect", and is the key to building up knowledge of our environment throughout our lifetime. The RL agent mimics the fundamental way in which humans (and animals alike) learn. As humans, we can perform actions and witness the results of these actions on the environment.

The "cause and effect" idea can be translated into the following steps for an RL agent:

1. The agent observes an input state
2. An action is determined by a decision making function (policy)
3. The action is performed
4. The agent receives a scalar reward or reinforcement from the environment Information about the reward given for that state / action pair is recorded

By performing actions, and observing the resulting reward, the policy used to determine the best action for a state can be fine-tuned. Eventually, if enough states are observed an optimal decision policy will be generated and we will have an agent that performs perfectly in that particular environment.

The RL agent learns by receiving a reward or reinforcement from its environment, without any form of supervision other than its own decision making policy. So, RL is a form of unsupervised learning. What this means is that an agent can learn by being set loose in its environment, without the need for specific training data to be generated and then used to teach the agent.

For more technical details about the overall RL process please read Chapter-4 of this report.

4.1 Reinforcement Learning(RL) Decision Engine for Offloading Process

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.

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.

The Idea

Let's consider the number of battery units consumed by our smartphone as our 'Reinforcements'. Now the idea is if we can minimize these units then we achieve reduction in the battery consumption. We need following things to train the Q function:

- action selection
- state transitions
- reinforcement received

The State of the function in our case holds parameters such as: Bandwidth, Data required by application, WiFi availability, CPU instance, Location of the device etc. The actions in this mechanism can be Offload, Do Not Offload, or Offload at a specific server. And finally reinforcements can be Battery Units consumed.

RL Offloading Algorithm:

State-Action Value Function as a Table

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.

State Values

In this scenario I have considered the parameters that I had used to create a fuzzy logic decision engine. Consider an application for which we want to train the Q function for various instances of following parameters:

- 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

This List of values that define State can be extended. We can add other Parameters to the List as follows:

- Location = Home, Office, Traveling
- Time = Morning, Afternoon, Night

So as you can see we can Define our state with relevant parameters.

Action Values

We need to train our Q function in all above mentioned conditions. In each such instance our smartphone will have three actions to choose from which are

- Local Processing
- Offload on Local Servers
- Offload on Remote Servers

We can add more options to this List, for instance If the smartphone is linked with multiple cloud vendors as demonstrated in [7], the smartphone will have to decide where to offload from multiple available clouds. After choosing any of these actions, the smartphone *system profiler* will record the battery units consumed during the processing of the application. The reinforcements will be assigned depending upon the battery consumption. The action which gave us best performance will be the one which consumed least battery units. So we are minimizing our battery units consumption here.

Representing the Q Table

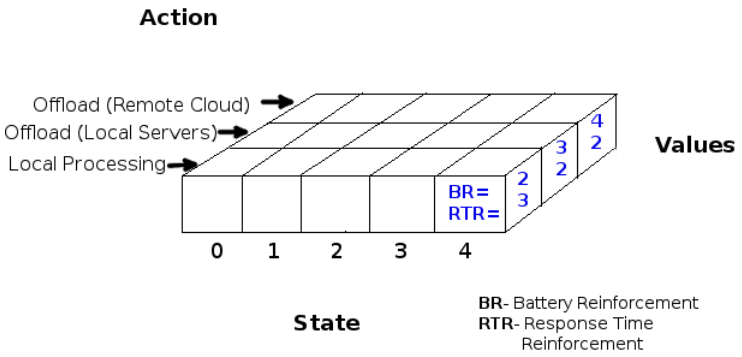


Figure 4.1: Representing the Q Table

The Agent-Environment Interaction Loop

For our agent to interact with its world, we implement the following steps:

1. Initialize Q (All the Reinforcement values are set to Zero)
2. Whenever the device changes its state (defined values in the state)
    - Run the application processing Locally and Record Reinforcement Values
    - Run the processing On Local Servers and Record Reinforcement Values
    - Run the application processing on Remote Cloud and Record Reinforcement Values

Significance of Reinforcements received

In the figure 4.2, I have demonstrated how the reinforcements are to be utilized to decide better performance in terms of battery units or response time. In the State 4 we can see that we received Reinforcement total of 5 when we the processing is done locally and when the decision engine chose to offload on Local Servers. But when the user demands energy savings at that time we will choose to not offload the app, whereas when the user requires better response time, we can allow the offloading system to be activated as the Reinforcement received by the system profiler is best for offloading on Local Servers. User can select his preference.

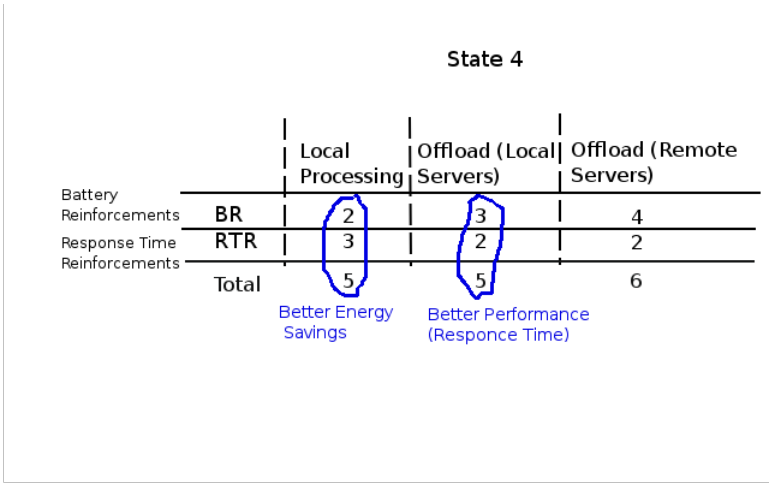


Figure 4.2: Reinforcements Received

Code

I have developed a Python code to demonstrate this algorithm. Let us look into some snippets of the code. In the function printParameters(board) shown below we can specify the parameters that we want to consider for training our Q function.

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

The epsilonGreedy function helps us choose our actions randomly in the initial trials and after sufficient trials we decay the value of epsilon to take the ‘greedy’ move that means to choose the action for which our Q function shows best reinforcement.

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

Here is how we assign values to our Q function. ‘$tuple(board2)$’ will give us some random values to our parameters Bandwidth, Data , CPU instance and so on. The digit after ‘$tuple(board2)$’ will specify the action taken, 0 for Local processing, 1 for Offloading on Local Servers and 2 for offloading on Remote servers. And the value assigned is the reinforcement received by that move, reinforcements here are 0, 1 and -1.

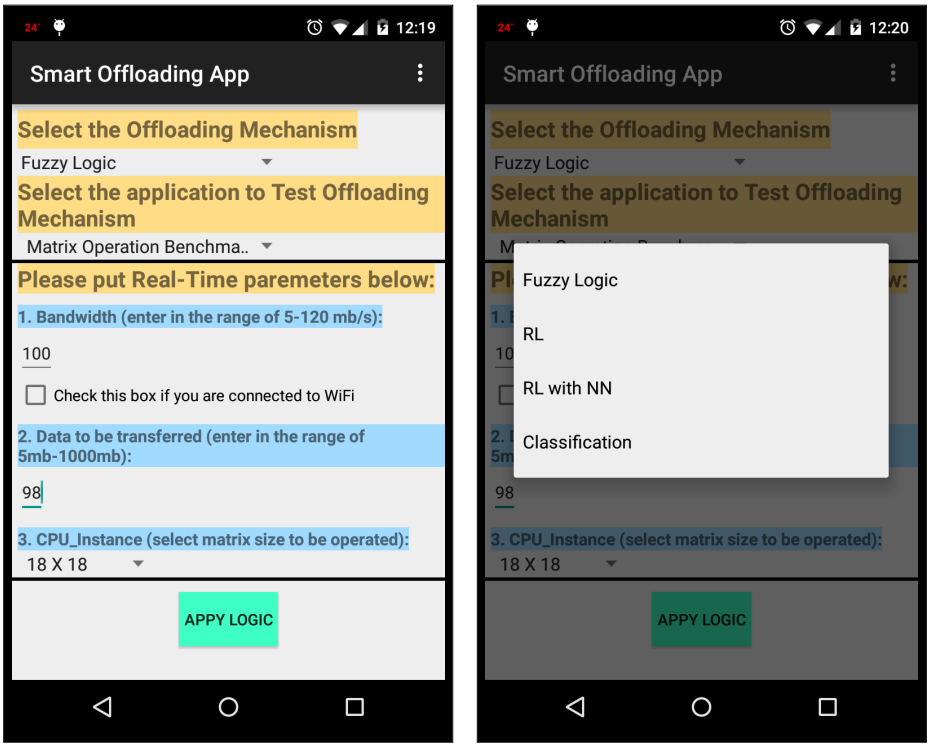
```
Q[(tuple(board2),1)] = 1
Q[(tuple(board2),2)] = 0
```

```
Q[(tuple(board2),0)] = -1
```

How the actual app will run ?

- The smartphone will have a training mode activated. So the training process will go on in the background. Whenever smartphone detects changes in some specified State parameters such as place, a time period, then it will run the algorithm. So that means it will do the app computation locally and subsequently on different cloud vendors. after it does that it will note down the key-value pair, in future while in similar situations this knowledge will be used.
- Suppose user do not want to run vigorous training mode which affects the smartphone performance, he can choose to deactivate the training mode. in such situation, whenever user uses a particular application it will go by default Local processing and if the smartphone finds second opportunity in that case it will try other options available, and trains itself. so this is in slow training mode.

We have developed a partially working smartphone app for the proof of concept, This decision Engine app is built in Android and shows all the algorithms presented in this work, user will have to manually select the type of algorithm and the application which he wanted to be offloaded when prompted by the decision engine, all the other parameters such as Bandwidth available, Data to be transfered can be customized by the user. To measure the bandwidth availability to the device we are using prebuilt smartphone app called nPerf [8]. In figure 4.3 we have shown some snapshots of the app.



(a) SmartOffloadingAppscreen (b) OffloadingAppOptionscreen

Figure 4.3: Smart Offloading App

4.2 RL with Neural Network

In this section I have used Neural Network with Reinforcement Learning to create a decision engine prototype for the offloading process. Reinforcement learning is learning by interacting with an environment.

I have developed a Neural Network code in Python and have demonstrated results produced by the prototype of Reinforcement Learning offloading app engine. I have explained how the offloading decision process can benefit from Reinforcement Learning and also presented different scenarios where it can be applicable in previous section.

Code

I have developed a Python code to demonstrate this algorithm. There are three important functions that I am using in this implementation which asre as follows:

```
def reinforcement(s,sn):
    if sn[0] < 4 and sn[1] == -1:
        r = 0
    elif sn[0] > 3 and sn[0] < 7 and sn[1] == 0:
        r = 0
    elif sn[0] > 6 and sn[1] == 1:
        r = 0
    else:
        r = -1
    return r

def initialState():
    initialStates = 0
    process = -1.0
    return np.array([initialStates, process])

def nextState(s,a):
    s = copy.copy(s)
    if s[0] < 10:
        s[0] += 1 #location
        s[1] = a
    return s

validActions = (-1,0,1)
```

The 'intialState()' function is where we define the starting point of our user, this state of smartphone can contain parameters like bandwidth, data requirement, CPU instance and others. The 'reinforcement(s,sn)' function gives reinforcements for the ideal situations in which parameters are suitable for either offloading or local processing. In the 'validActions' we can see three choices which are as follows:

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

In the figure 4.4 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 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

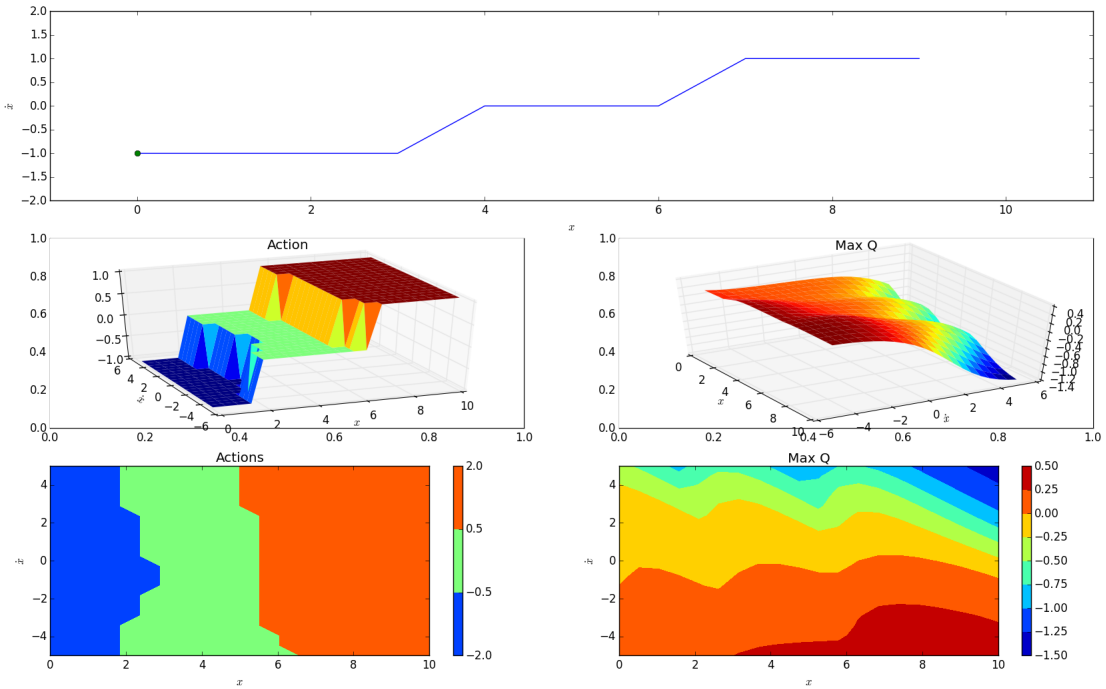


Figure 4.4: Best Action: Local Processing

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 5

Smart Offloading Decision Engines

In this section let us see some of the important Related Work which are focusing on developing a decision engine for the smartphone which ultimately takes the correct offloading decision.

5.1 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 [9] 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. I have also demonstrated the working of the app. Java code that was developed for the app is shown in the end. In this section a Fuzzy Logic Decision Engine Implementation is discussed.

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: 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).

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.

Following are the Fuzzy sets and Some of the rules considered:

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 I am assuming there are multiple locations available with the user where he can offload his application processing and data. After defining these parameters I have assigned grade of truth values to each of the set considered which fuzzy logic uses to classify the outputs.

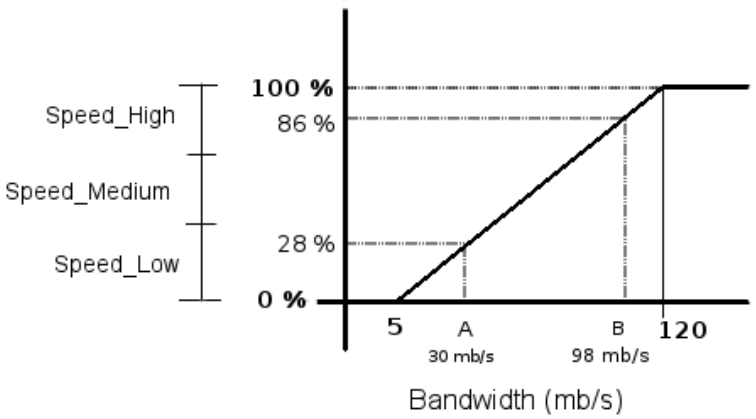


Figure 5.1: Best Action: Local Processing

Some of the 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

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.

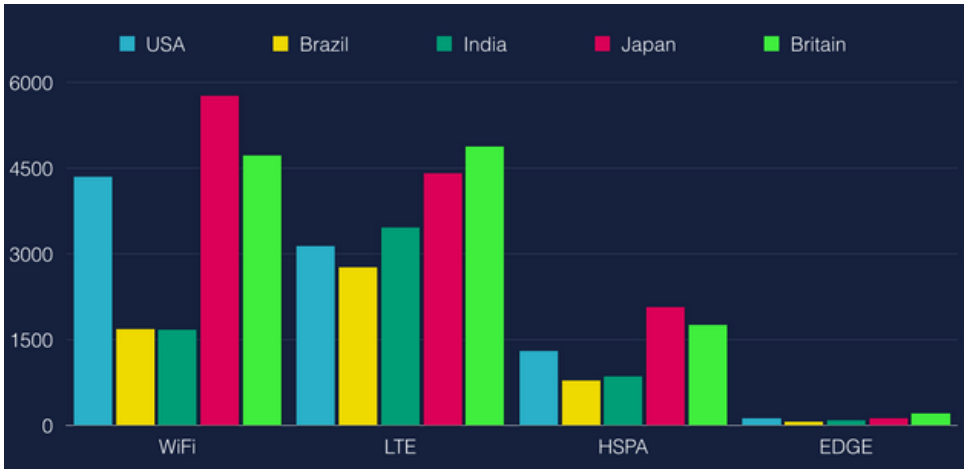


Figure 5.2: Varying Internet Speed across different Countries (Source: Facebook code Community)

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 5.2 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.

Because of such variations in the different technologies around the world, it is difficult to rely on the fuzzy logic decision engine presented in [9]. This is because the app developers will have to customize the decision engines depending upon which part of the world the device lies.

5.2 Cuckoo

Cuckoo [10] is a computation offloading framework that targets the Android platform; it offers a simple programming model that supports local and remote execution. Cuckoo offloading mechanism takes the offloading decision based on the server availability only, without considering other contextual information.

5.3 Smartphone Energizer (SE)

Smartphone Energizer [11] 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 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.

Chapter 6

Experiments and Results

For evaluation of our Offloading methodology and comparison with other similar works we have used three pre-built smartphone applications. The comparative analysis of these three applications can be see in figure 6.1

Figure 6.1: Energy Consumption : Local Processing Vs Offloading on Cloud

The Sorting Benchmark app is a simple app which sorts numbers and measures the time consumed while we sort those numbers. The Matrix Benchmark Application can perform matrix operations such as calculating Inverse of a matrix, determinant of a matrix and other operations. User have to manually input the size and elements of matrix and choose which operation he wants to perform. In these two benchmark applications the data that need to be transferred to Cloud in case of offloaded processing is very low. In our third application which is a Health monitoring app, the data to be transferred is larger. The processing of these three apps will be done either locally or on the cloud as directed by the Decision engines. In figure 6.1 we have given a comparison of energy consumption while Local Processing and Offloaded Processing.

In figure 6.2 we have shown the output of our RL based Decision Engine app at different location on the CSU Oval campus. We can see that due to varying parameters such as Network Bandwidth availability, Wifi connectivity the decision of offloading Engine varies.

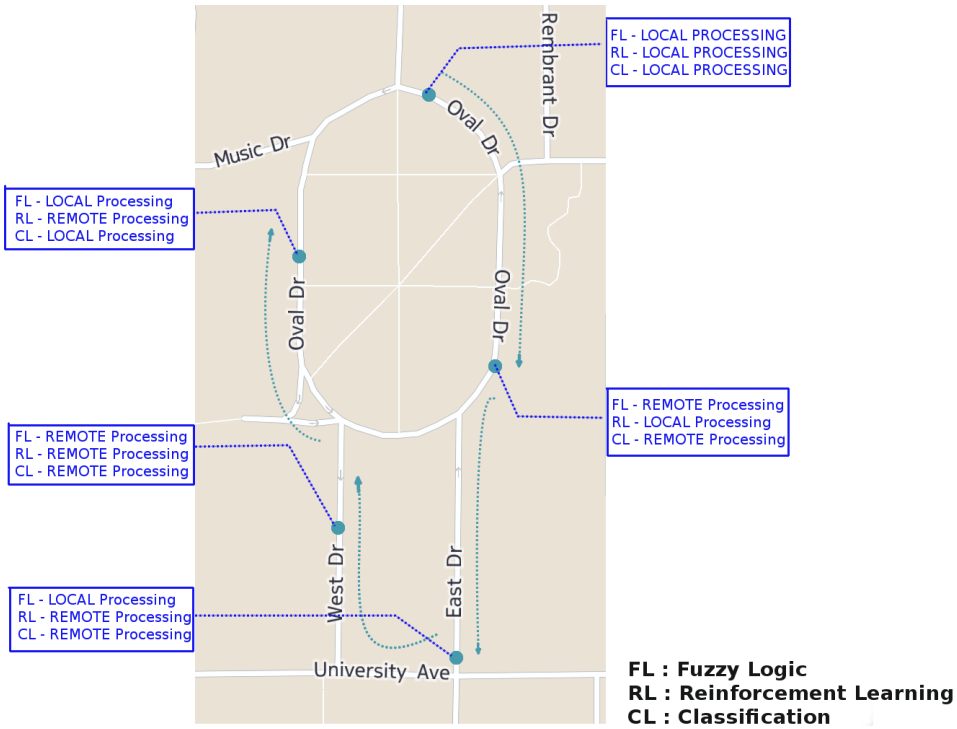


Figure 6.2: Decision Engine Outputs while Real Time Usage of Offloading App

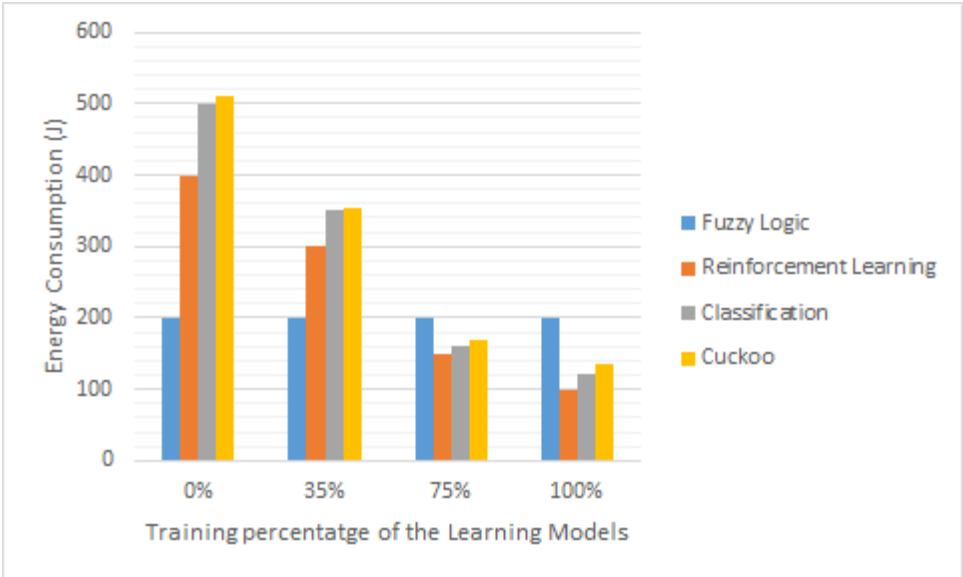


Figure 6.3: Energy Consumption by Different Models with Varying percentage of Training for Matrix Operation Benchmark App

Chapter 7

Discussion and Future Work

Does this work and other offloading system make cloud computing the ultimate solution to the energy problem for mobile devices? Not quite. While cloud computing has tremendous potential to save energy, designers must consider several issues including privacy and security, reliability, and handling real-time data [1].

What is the recent trend in Smartphone app and Cloud Industry which encourages offloading services

Offloading system is far from adapted, but the leading Cloud technology firms such as Amazon have launched many services to encourage compute offloading on Cloud, for example Amazon Elastic Compute Cloud (EC2) and Lambda. This year Amazon has launched 'AWS Lambda', a compute service that can run our code in response to events (such as upload of an image, or a sensor output) and automatically manages the compute resources. AWS Lambda starts running the code within milliseconds of an event such as an image upload, in-app activity, website click, or output from a connected device which means better response time performance. Also with most of these Cloud services you pay only for the requests served and the compute time required to run our code. Billing is metered in increments of 100 milliseconds, making it cost-effective and easy to scale automatically from a few requests per day to thousands per second [12]. Also there are similar options are emerging such as StackStorm, it's an open source software project that is typically runs on premise or by users on their public clouds, whereas Lambda is of course an AWS service [13].

What kind of applications can benefit from this work

Gaming applications such as Chess, where amount of Computation required is huge, and the Data that need to be transferred is less.

Image processing applications, which can readily process the images that are already present in the cloud. For instance if we want to create thumbnails of series of images from our albums, offloading of such process makes sense as the image data most likely will already be present in the cloud, if the user is using Cloud services such as Google's Unlimited Image storage.

Machine learning algorithms such as Reinforcement Learning with Neural Network and others requires multiple trials for a well trained Q functions which might run into hundreds or thousands of iterations. For the compute intensive applications the offloading is beneficial as we have seen in [1]. For example if we want to train a neural network for a smartphone device which has thousands of iterations, we can offload the training of Q function on the Cloud, and use the trained function for the decision making.

Machine learning techniques can be really useful for the smartphone applications. The offloading domain research can benefit from classification or reinforcement learning. I have demonstrated some ways with which we can enhance the offloading process with the help of an offloading engine which uses machine

learning techniques. I found Reinforcement learning is very exciting and promising when used in intuitive situations such as the one which is presented in this work where we can save battery power. The main obstacle while using machine learning for the smartphone devices can be to achieve real time processing for the dynamic applications. When the neural network needs a lot of processing to train itself at that time the smartphone-cloud coupling can prove useful as the device need not bother about big number of iterations, it can just offload the processing on cloud.

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] H. Flores, S. N. Srirama, and C. Paniagua, "A generic middleware framework for handling process intensive hybrid cloud services from mobiles," in *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, pp. 87–94, ACM, 2011.

[8] "nperf from google play." `https://play.google.com/store/apps/details?id=com.nperf.tester&hl=en`.

[9] 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.

[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] 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.

[12] "Aws lambda." `http://aws.amazon.com/lambda/`.

[13] "Stackstorm." `http://stackstorm.com/product/`.