Project Report
Machine Learning techniques for
Offloading Smartphone Computing on Cloud to Save battery Power

Aditya Khune

December 15, 2014

Contents

Abstract

Smartphones have become the primary computing platform for many users. Various studies have identified longer battery lifetime as the most desired feature of such systems. Low-power design has been an active research topic for many years. Some studies such as in [1] suggest that many applications are too computation and if offloaded on the cloud then can actually save battery power.

In this project I have studied this solution of offloading the computing on the cloud. After studying the problem I have proposed a decision engine based on some machine learning techniques which will act as an enhancement to the offloading process. In [2] authors have suggested a similar decision engine based on fuzzy logic. I have implemented a similar decision engine to start with and then used Reinforcement learning to take the right decision of whether to offload or not.

1 Introduction

Mobile cloud computing is arising as a prominent domain that is seeking to bring the massive advantages of the cloud to the resource constrained smartphones. Offloading Smartphone Computing on Cloud is a technique that allows to empower the computational capabilities of mobiles with cloud resources. Code offloading refers to a technique, in which resource intensive mobile components are identified and offloaded to remote processing in order to be executed by cloud-based surrogates. Most of the prominent works in the domain have proposed solutions to overcome the issues related with deciding whether to offload or not to cloud.

1.1 Offloading computation to save energy

Cloud vendors provide computing cycles, and users can use these cycles to reduce the amounts of computation on mobile systems and save energy. Thus, cloud computing can save energy for mobile users through computation offloading. Virtualization, a fundamental feature in cloud computing, lets applications from different customers run on different virtual machines, thereby providing separation and protection.

In the paper [1] the authors have done an Energy analysis for computation offloading, and have given a mathematical explanation as follows:

$$B_o \approx constant \times \frac{D}{C}$$

where B_o is the minimum bandwidth required for offloading to save energy, determined by the ratio of $(\frac{D}{C})$. If $(\frac{D}{C})$ is low, then offloading can save energy. Thus, as Figure 1 shows, offloading is beneficial when large amounts of computation C are needed with relatively small amounts of communication D.

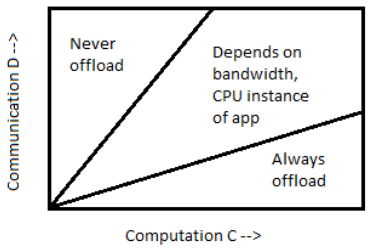

Figure 1: Offloading decision Communication vs Computation

1.2 Offloading Decision Engine

In this project I have demonstrated some strategies to enrich the offloading decision process with machine learning methods. The decision engine is the mechanism which takes the decision of offloading for the device depending upon 'contextual' information of the device such as the bandwidth.

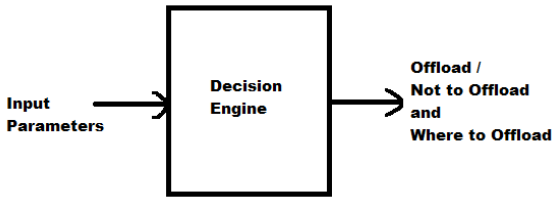

Figure 2: Offloading Decision Engine

Apart from bandwidth other parameters that can be considered are the CPU instance compute intensity(CPU_low/CPU_normal/CPU_high), Data requirement of the application (data_small/data_medium/data_big).

In the next section I will discuss the techniques that I have used and then we will study the subsequent results obtained. I have used following methods to create this mechanism:

- Reinforcement Learning
- Fuzzy Logic
- Classification

2 Implementation and Results

In this section I have described all the methods that I have used for the offloading engine mechanism. The Reinforcement learning algorithm is implemented in Python. I have modified the reinforcement learning and neural network code that was provided in the class CS 545 by Dr. Anderson. The fuzzy logic engine is implemented in Java with Android Software Development Kit (SDK). And for classification I have used the same code of Least Squares Classification that was provided in the notebook by Dr. Anderson.

2.1 Reinforcement Learning

2.1.1 The Concept

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.

We have studied in the class that the main objective of Reinforcement learning is to make this approximation as accurate as possible:

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

here s_t = state, a_t = actions, and r_t = reinforcements received. This is usually formulated as the least squares objective:

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

2.1.2 The Idea

Now let's consider the number of battery units consumed by our smartphone as our reinforcements. If we can minimize these units then we achieve reduction in the battery consumption! very intuitive right?

We need following things to train the Q function:

- action selection
- state transitions
- reinforcement received

The State of the function in our case can have two parameters: Location, Battery Units consumed. The two actions in this mechanism should be Offload, Do Not Offload.

We are using Neural Network as our Q function. We will have to evaluate each state and action pair to decide which action to take next. Here is how we train our neural network function:

$$nnet = nn.NeuralNetworkQ(ni, nh, output, ((p1, p2), (b1, b2), (a1, a2)))$$

Here, ni=number of inputs, nh=number of hidden units, output=number of output, p1-p2= range of locations, b1-b2= battery units range, a1,a2=actions to take.

Here is some code of the main functions of the implementation:

```
def initialState():
    position = np.random.randint(1,5)
    processing = np.random.randint(0,2)
```

```
    energy_units = 1
    return np.array([position, processing, energy_units])

def reinforcement(s,s1):
    return -1 if(s1[2] > s[2]) else 0

def nextState(s,a):
    s = copy.copy(s) # s[0] is position, s[1] is velocity. a is -1, 0 or 1
    s[2] = np.random.randint(1,5)
    if (s[1] == 0) :      # Bound next position. If at limits, set velocity to 0.
        s[1] = 1
    elif (s[1] == 1) :
        s[1] = 0
    return s
```

In Figure 3 we can see the contour and surface plots of our trained Q function. I am using 5 different locations for the device, with serial numbers 0 to 5. In 'Actions' plot we can see the actions taken by the Q function, it shows that it opts to do a local processing(-1) when the device is at locations 0 & 1; and opts for offloading of processing (1) when at location 2,3 and 4.

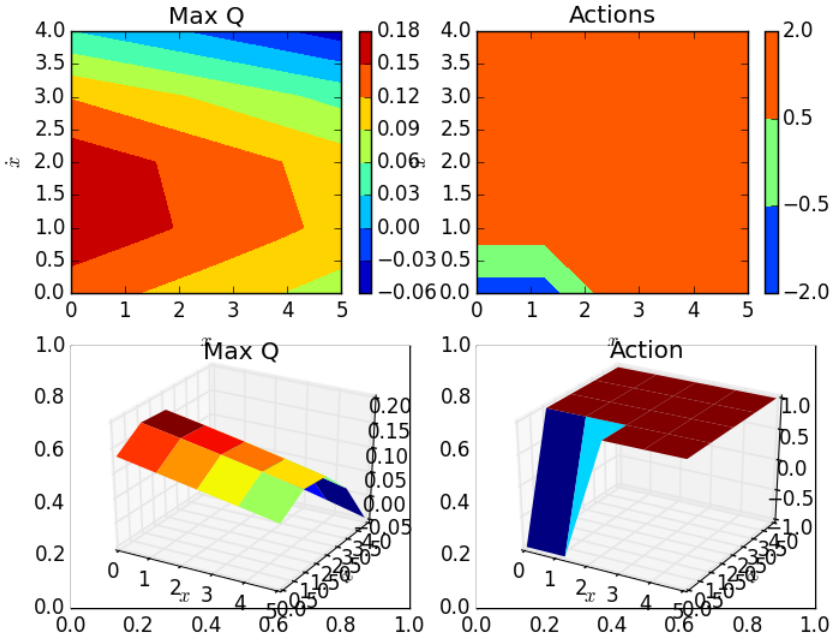

Figure 3: Trained Q function plot

2.2 Fuzzy Logic Decision Engine

In this section I have explained the Offloading Decision Engine smartphone app that I have created for an Android device. In [2] the authors have proposed a fuzzy decision engine for code offloading, that considers both mobile and cloud variables. I have implemented a similar engine with relevant parameters and with slightly different rules. I have also demonstrated the working of the app. I have used Android Studio platform to develop this app for the Google Nexus 4 smartphone on my Linux machine.

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

Some of the Rules considered

- Remote Processing = Speed_High AND Data_Small AND CPU_Normal
- Remote Processing = Speed_Low AND Data_Small AND CPU_High
- Remote Processing = Normal_Low 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

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.

Now let us have a look on the user interface of our offloading app in Figure 4. As I have said in my

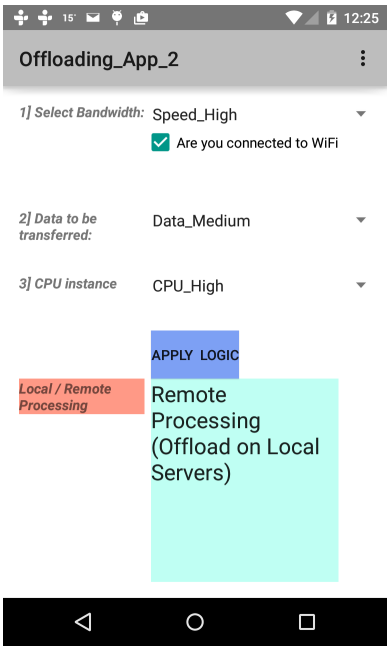

Figure 4: Offloading Decision Engine

current implementation all these parameters are to be filled by user manually. I have created a single screen

app where the user can choose from the available options, the Bandwidth available, Data to be transferred, CPU instance requirement of the app, is WiFi available with the user, etc. After giving all these inputs we click on the 'APPLY LOGIC' button which can be seen in blue color. The decision of the engine will be seen in the Big Output Box on the right bottom corner.

In Figure 5 I have plotted all different parameters that are considered to decide about the processing offloading. The plot for 'processing' is the output our decision engine provides. In this plot when the 'processing' is at zero the output of the decision engine is 'Local Processing', when the processing is at 10000 the output is 'Offload on Local Servers', when the processing is at 20000 the output is 'Offload on Remote Servers'. The decision of offloading is generally seen when the 'CPU instance' is on Higher side and when the 'Data' used by the app is on lower side.

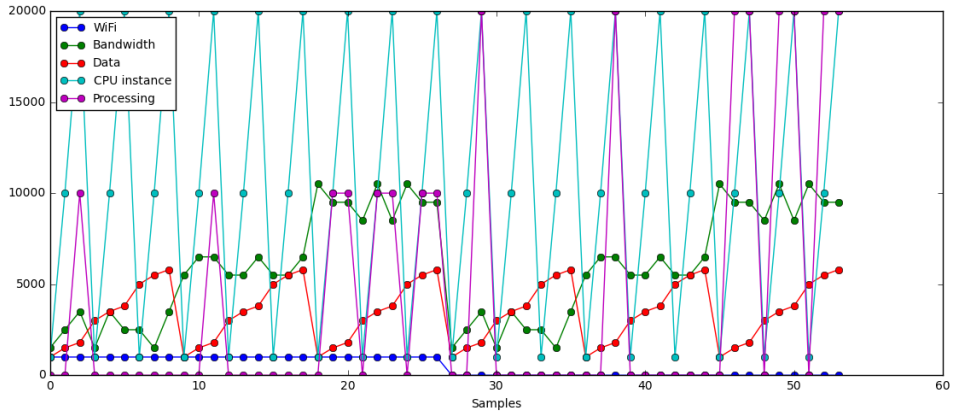

Figure 5: Offloading Decisions with all parameters

2.3 Classification

Finally, I have also tried using classification with Least Squares algorithm. I have created a sample data which looks as follows:

WiFi	Bandwidth	Data	CPU	Processing
1	1500	1000	1	0
0	2500	1500	1	0
1	3500	1800	2	1

Here the value WiFi is 1 when it is available and 0 when it is unavailable, values of Bandwidth and Data are in kbs, CPU instance 1 when low and 2 when high, and processing local when value is 0 and remote when value is 1. So we will be predicting the 'Processing' column with the help of classification function. With some 60 Samples I have divided some data as testing and training data. I have used the code that is provided by Dr. Anderson in the class for the Least Squares algorithm [3].

In Figure 6 we can see the Actual and predicted values of the offloading decision.

Because I have very less samples with me I was not able to provide enough training data set, which resulted in poor Percentage of correctness of decision values.

3 Discussion

For the compute intensive applications the offloading is beneficial. For example if we want to train a neural network for a smartphone device which has thousands of iterations, we can offload he training of Q function on the Cloud, and use the trained function for the decision making.

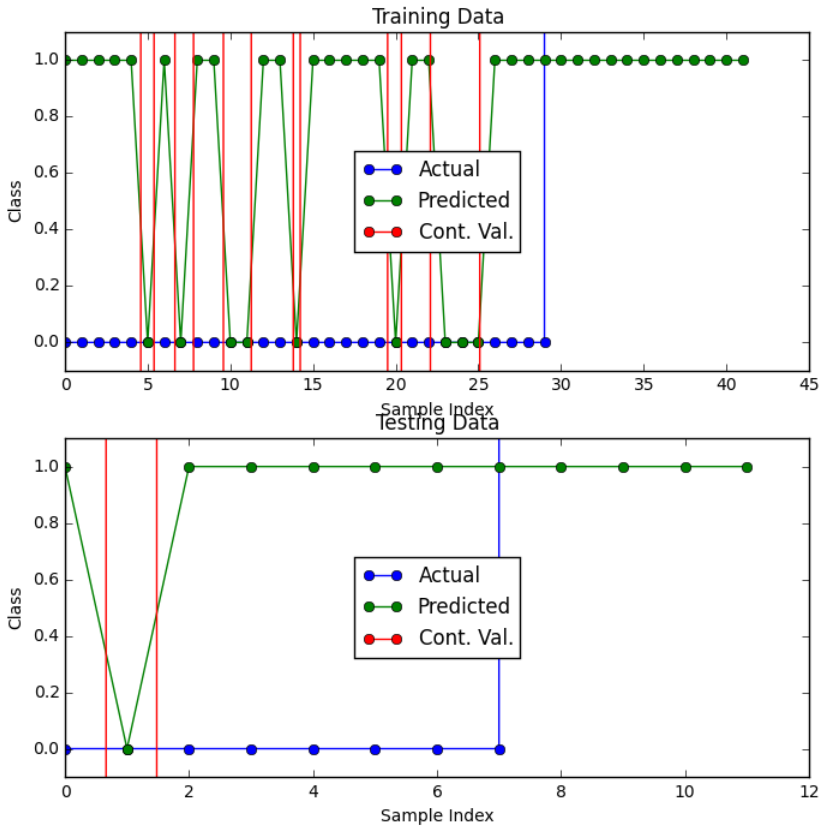

Figure 6: Plot of Least Squares Classifiaction for Training and Testing data
Percetage Correct:Training: 56 % Testing: 20 %

Does this 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].

4 Conclusion

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 project where we can save battery power. We can get good results with the classification algorithms as well when we have large training data set available with us.

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.

References

[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. 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.

[3] C. Anderson, *Cs545 machine learning lectures*. Colorado State University, 2014.