International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 5, No.1 (Jan-2016)

http://dx.doi.org/10.12785/ijcds/050103

Efficient Real-Time Mobile Computation in the Cloud Using Containers

S M Azharul Karim[1], John J Prevost[2] and Paul Rad[3]

[1]*Department of Electrical Engineering, University of Texas at San Antonio, San Antonio, USA*
[2] *Assistant Professor, Department of Electrical Engineering, University of Texas at San Antonio, San Antonio, USA*
[3] *Chief Research Office and Assist Director, Open Cloud Institute, University of Texas at San Antonio, San Antonio, USA*

Received 14 May 2015, Revised 17 July 2015, Accepted 2 Oct. 2015, Published 1 Jan. 2016

Abstract: Mobile devices have limited resources in terms of power and bandwidth. Cloud computing offers a way to reduce the power consumption of mobile devices by offloading computation to the cloud. However, offloading computation means an increase in communication energy consumption. The trade-off between energy and network characteristics (bandwidth/latency) in a mobile device is very important. Therefore computation offloading must be done strategically. The optimum utilization of the available mobile device resources needs to be assured. In this paper, we propose an intelligent and dynamic algorithm to offload computation to the cloud utilizing containers. We focus on offloading computation based upon the communication topology, device energy and user inputs. We analyze the cost of offloading computation for different user inputs, and based on the inputs, we decide whether to offload the application to the cloud or not. Our algorithm was implemented using two different approaches, a fuzzy-logic system and a neuro-fuzzy system. These were both simulated using MATLAB®, and the results compared. Our previous work demonstrated that the fuzzy-logic system performed better than the referenced system. This paper demonstrates that the neuro-fuzzy approach, utilizing a container based hosting model, improves on our prior results.

Keywords: Cloud computing, offloading computation, network topology, bandwidth, latency, MATLAB®, Container.

1. [1]INTRODUCTION

Mobile cloud computing is becoming more and more popular. Every day, billions of devices vie for access the cloud. With the advancement in wireless connectivity technologies (4G, 4G+, 5G, Wi-Fi), and with the plethora of cloud services available, connecting to the cloud has become very easy. Even a few years ago, one had to pay a lot of money to buy a smart device with Internet access. Today, smart devices, such as Mobile phones, smart watches, tablet PCs, etc are all becoming less expensive. Devices like Apple iPhone [1], Samsung Galaxy phones [2], Microsoft Lumia phones [3] etc are becoming cheaper and popular. More and more people are using them to access the Internet and the cloud. According to Cisco [4], global mobile data traffic grew 69% in 2014. Total number of mobile devices in 2014 grew to 7.4 billion, compared to 6.9 billion in 2013. Smart phones accounted for 88% of that growth. And the number of mobile devices will continue to grow.

As companies are capitalizing on the proliferation of mobile and wearable devices to develop new products and services globally, they rely on cloud capabilities to fulfil their large-scale parallel computation requirements. Many data analytic applications are too computation intensive to perform on a mobile or wearable device. If a mobile user wants to use such applications, computation must be performed in the cloud. Figure 1 shows us the mobile cloud distribution model.

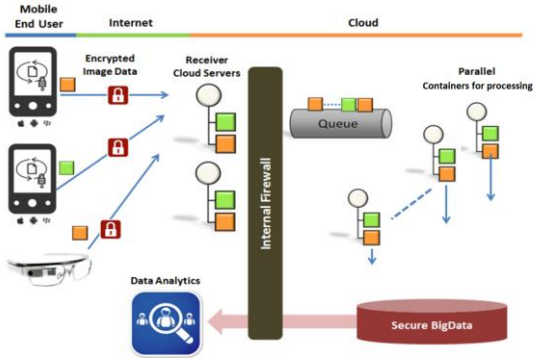

Figure 1: Mobile-Cloud Distributed Computing Model

[1] This research is partially supported by the Open Cloud Institute at the University of Texas at San Antonio.

E-mail: s.m.azharul.karim@gmail.com, jeff.prevost@utsa.edu, paul.rad@utsa.edu

Cloud computing is a new paradigm where application services are provided through the Internet and computing resources such as processing, memory, and storage are not physically present at the user's location. Cloud computing can enhance the computing capability of mobile systems, but requires higher application mobility [17] [18].

Although mobile devices are becoming smarter and more powerful, they have limitations. Smart-phones are energy hungry. The high-end devices can often run apps smoothly, but they also require a lot of energy to operate. According to an article in Science Time [5], an iPhone requires as much energy as a refrigerator. The battery power has not increased significantly in recent years. However, people want better batteries on their smart-phones that can run their devices for more than 24 hours. In a mobile device, different applications consume energy in different ways [9]. There are different approaches to reduce mobile energy usage. One way is to calculate how much power the application requires before it is executed [6]. In [6], the authors suggest an algorithm that can calculate energy consumption of an application by examining the source code of that application. Another way to optimize energy consumption is to offload computation to the cloud [7][8]. This approach reduces mobile energy consumption significantly but it doesn't take into account the available network and environment. According to [10], cloud mobile augmentation allows resource-rich clouds to increase, enhance, and optimize computing capabilities of mobile devices aiming at execution of resource-intensive mobile applications. However, an increase in offloading to the cloud means the device has to transfer more data to the cloud. Offloading more data and computation increases the energy to communicate with the cloud. If communication energy is more than executing the computations in the phone, then offloading computation may not be a wise decision. So, a smarter approach is required to offload computation to the cloud. In [11], the authors suggest an algorithm to offload computation to the cloud by partitioning every application into different processes. And then the device will offload computation heavy processes to the cloud, and execute data-intense processes in the device. That way it can ensure better optimization of mobile device resources. But this approach does not consider the wireless broadband service available (3G, 4G, 4G+, and Wi-Fi). The mobile communication energy may vary based on different networks and their data rates.

In this paper, we have focused on offloading applications to the cloud based on the available network and its characteristics, device energy and user inputs. We have applied fuzzy logic and adaptive neuro-fuzzy modeling to decide whether to offload the application to the cloud or not. We have implemented our algorithm separately using both systems. Using our algorithm, we have implemented an image matching application in MATLAB®. We have allowed for different input parameters to be used to decide where to execute the application. In this paper, we have shown that through offloading applications to the cloud, our algorithm saves device energy. We have compared our result with [11], and showed that our algorithm saves more time compared to the approach in [11]. We have also compared the two systems to show which system performs better for the same inputs. Figure 2 represents the implemented algorithm.

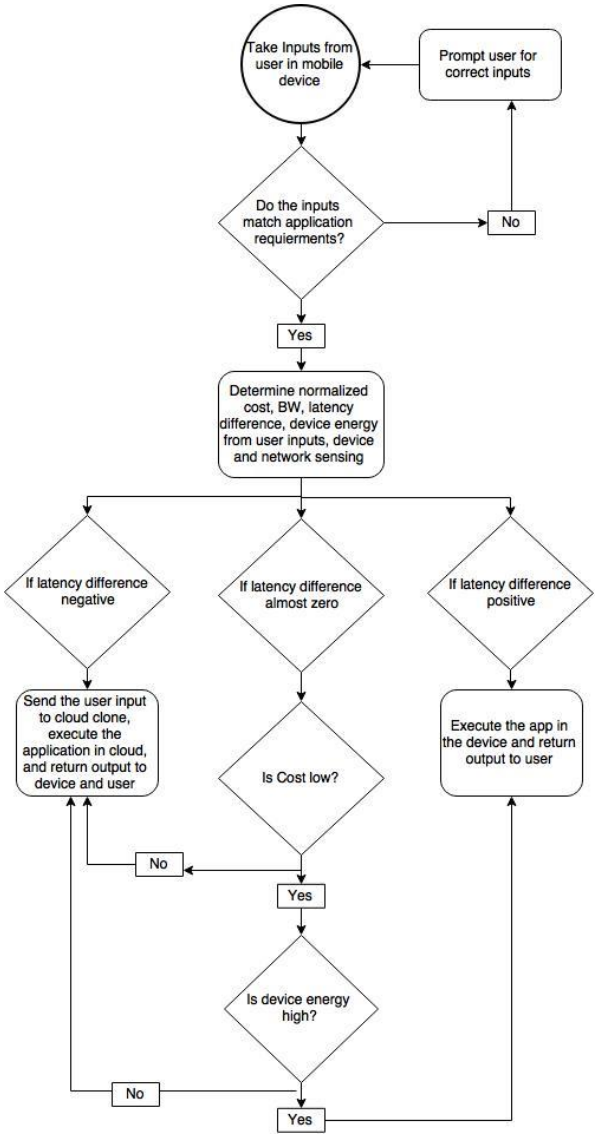

Figure 2: Flow diagram of implemented algorithm

The rest of this paper is organized as follows. At first we will describe the system model used in our algorithm. Next we will implement our model using a simulated numerical example. We then evaluate our results and lastly provide a conclusion of our work.

2. System Model

We will first assume access to a cloud infrastructure with unlimited computational ability and infinite power supply. In this cloud, we will implement our device clone that will be used to execute different applications. Our objective is to optimize energy use in the device.

For that, we consider the following parameters- (1) available bandwidth, (2) energy level on the device, (3) user input data and (4) latency incurred for executing each application in cloud. Based on these parameters, we have decided whether to offload the application to the cloud or not.

This decision-making is done by two different approaches: A. Fuzzy logic, B. Adaptive neuro fuzzy modeling. Both approaches implement the same algorithm. For the fuzzy logic approach, we have to design the membership functions, rules, and variables. On the other hand, the neuro-fuzzy analysis takes a training data set as input, and creates a fuzzy transfer function using that training data set.

Figure 1 represents the flow diagram of the algorithm that we have implemented. It shows how the algorithm decides about offloading to the cloud based on the above mentioned parameters.

To decide whether to offload to cloud or not, we have made some assumptions. Table 1 represents the assumptions that we made for the paper.

Table 1: Assumptions made for this paper

#	Description of Assumption
1	Combined file size of the two input images will not exceed 10 MB.
2	The two images will not have combined dimension of more than 50,000,000 pixels.
3	Energy level on the handheld device ranges between 0-100
4	Both the device and the cloud clone has the same application, and the application can be executed in the device or in cloud clone.
5	We assume our handheld device has an ARM Cortex A7 processor, and the cloud computers have Intel Xeon 5300 series processor.
6	For each iteration, execution time in ARM processor is 1ns.
7	For each iteration, energy consumed in ARM processor is 10 µW.

(Note: Assumption 1 and 2 are the constraints of the image comparing application. The application can handle images with even bigger file sizes and dimensions, but for the ease of simulation, we have constrained the inputs to the above-mentioned values. These assumptions are considered for both the fuzzy and the neuro-fuzzy analysis.)

A. Lightweight Containeration of Applications

Virtual machine (VM) based virtual infrastructure has been adopted widely in cloud computing environment for elastic resource provisioning. Performing resource management using VMs, however, is a heavyweight task [19]. Container images, such as those prepared via the Docker, or Rocket projects [20], are rapidly becoming the unit of deployment for cloud-native application architectures. Containers leverage modern Linux kernel primitives such as control groups (cgroups) and namespaces to provide resource allocation and isolation features as those provided by virtual machines with much less overhead and much greater portability. Figure 3 shows the containerization of mobile apps.

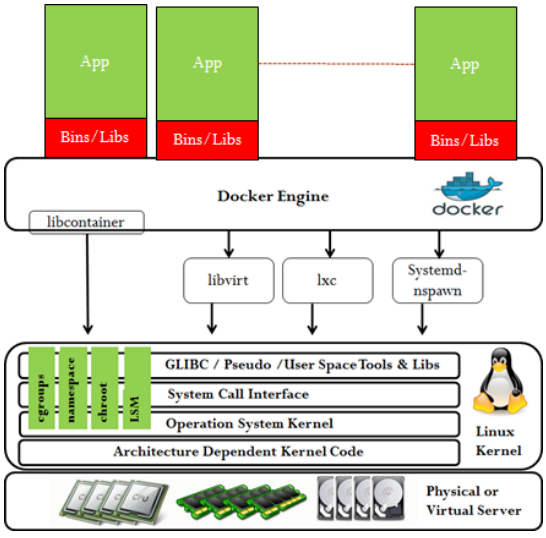

Figure 3: Containerization of Mobile Applications

The experiment results show that our proposed containerized based resource management approach outperforms the VM-based approach in terms of feasibility and resource-efficiency [18].

B. Fuzzy variables

1) Cost (execution energy)

Mobile applications require a lot of energy to execute. With the increase in complexity and size of mobile apps, modern mobile apps consume a lot of energy. That is why energy required to execute the applications should be considered before we offload applications to cloud. We have defined energy required to execute the application as "Cost". We have assumed that the cost is proportional to number of iterations in the application. For the image matching application, the number of iterations is equal to the user input image file dimensions.

Let us assume, Image 1 dimensions are X1 and Y1, and image 2 dimensions are X2 and Y2.
Then,

$$\text{cost} = \frac{X1 \times Y1 + X2 \times Y2}{50000000} \quad \textbf{(1)}$$

We have categorized cost in three groups: low, medium and high. Figure 4 shows us the three normalized costs.

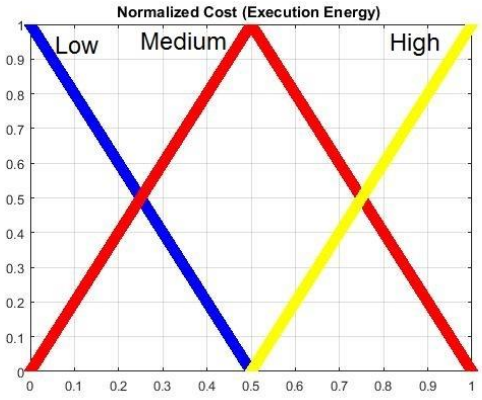

Figure 4: Normalized Cost (execution energy)

2) Device Energy

Our handheld devices have limited energy. We have to consider our device energy level before we want to upload any application in the cloud. We have categorized energy level on the device as: low, medium and high. Figure 5 represents the three device energies.

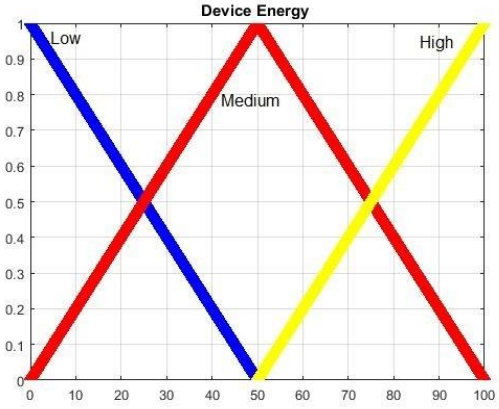

Figure 5: Device energy

3) Latency Difference

Latency is the delay between user input and output. High latency is not desired by anyone. That's why we have given it the highest priority. We have measured execution time in the device, and execution time in the cloud. There are two types of latencies in our system:

a. Latency for executing the application in the device
b. Latency for executing the application in the cloud clone.

According to [12], clock speed of an Intel® Xeon® 5300 series processor is 3.0 GHz. Also according to [13], clock speed of an Arm Cortex-A7 processor is 1.5GHz. Although only clock speed is not sufficient to determine the speed of each processor, for simplicity we will assume that each CPU performance is proportional to their clock speed. So we define the ratio between these two processors as a factor by which Intel® Xeon® 5300 series processor is faster than ARM cortex A7 processor. We define our variable as:

$$Xeon/ARM = \frac{3GHz}{1.5GHz} = 2 \quad \textbf{(2)}$$

This factor will be used to calculate execution time in cloud clone.
We assume, total number of iterations for the given inputs as N.
Now, we define the two latencies in detail:

$$Latency\ for\ device = Time\ to\ execute\ application\ in\ device\ (ARM\ Cortex\ A7\ processor)$$

Which can be re-written as:

$$Latency\ for\ device = N \times 1ns \quad \textbf{(3)}$$

And,

$$Latency\ for\ cloud = Time\ to\ execute\ application\ in\ cloud(Intel\ XEON\ processor) + Latency\ of\ transmitting\ and\ receiving\ data(communication\ time)$$

Now, latency to execute the application in Cloud

$$\text{Execution time in cloud} = \frac{N \times 1ns}{Xeon/ARM} \quad \textbf{(4)}$$

To calculate the communication time between mobile device and the cloud clone, we define a variable "Bandwidth". Bandwidth defines available data rate of the network. The higher the data rate, the more data we can send through the network. In the simulation, bandwidth is used to measure the latency of the network for the given input file sizes. Bandwidth is selected randomly in the range of 0.01 to 1, where 0.01 represents lowest possible bandwidth and 1 represents maximum bandwidth.

To calculate the latency to transmit and receive data from the device to the cloud clone, we have defined a Gaussian distribution. According to [14], the maximum latency for 4G LTE network is 150 ns. We have set the maximum value of the Gaussian distribution as 150, and the minimum value as 0. From the Gaussian distribution,

we select the time to transmit, or receive data. We define this latency as, "network latency".

Figure 6 shows us the Gaussian distribution from which we have selected the network latency.

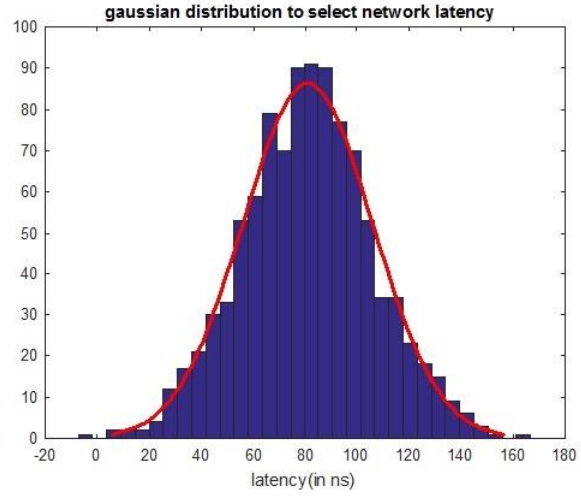

Figure 6: Gaussian distribution for network latency

Although this latency value is fixed, the time to transmit the total data might be even more. If the available bandwidth is low, then the device might need to split the data and then send them one by one. This will require more time than sending the data at once. To calculate this effect of low bandwidth, we have read the total size of the two input files in Bytes and stored them in a variable called "*filesize*". According to [15], the average upload data rate for Verizon 4G LTE network is 14.63 MB. We read the normalized available bandwidth in the "*BW*" variable.

Time to transmit the data to cloud is defined as:

$$Tx\ time = \frac{network\ latency \times 1ns \times filesize}{BW \times 14.63 \times 1048576} \quad \textbf{(5)}$$

The output of the application is a floating-point value. In any 4G LTE network, it will need one unit time, or "network latency" time to send the data back to the device.

Time to receive the data from cloud,

$$Rx\ time = network\ latency \times 1ns \quad \textbf{(6)}$$

Then, the equation to calculate latency for cloud becomes-

$$Latency\ for\ cloud = \frac{N \times 1ns}{Xeon/ARM} + \frac{network\ latency \times 1ns \times filesize}{BW \times 14.63 \times 1048576} + network\ latency \times 1ns$$

Finally, we calculate the difference between cloud latency and device latency, and use it as a fuzzy input set.

$$Latency\ difference = Latency\ for\ cloud - Latency\ for\ device$$

If the Latency difference is positive, it means that the latency for the cloud is greater than that of the device, and therefore the application should be executed in the device. If it is negative, it means the application should be executed in the cloud. If it is zero (or close to zero) other input parameters will come into consideration and based on these parameters, a decision will be made regarding the run location of the application. Figure 7 represents the three latency differences.

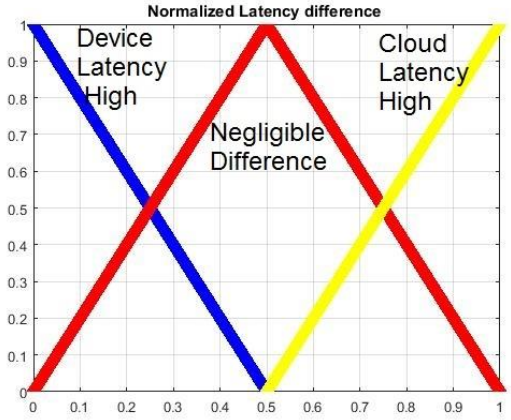

Figure 7: Latency difference

4) Decision

We have divided the output based on where to execute the app: in the device, or in the cloud. Based on the inputs, our algorithm generates an output. The output has two possible outcomes- a. execute in device, b. execute in cloud. Based on the decision, the application is executed either in the device or in the cloud. If it is executed in the cloud, the data is sent from the mobile device to the cloud, and after execution, the result is sent back to the device. Figure 8 represents the output decisions.

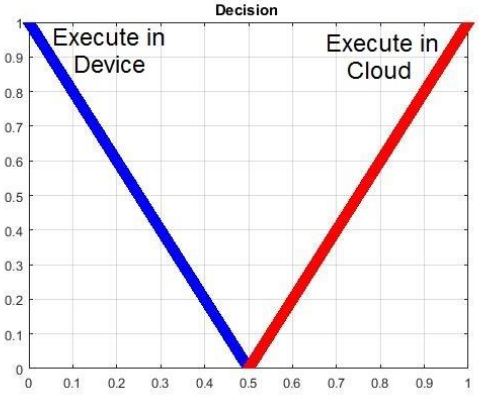

Figure 8: Output decision

C. Adaptive Neuro-Fuzzy Modeling

For the neuro-fuzzy approach, we have used the "Neuro Fuzzy Designer" toolbox in MATLAB®. The neuro-fuzzy designer toolbox combines artificial neural network and fuzzy logic to implement human-like reasoning in computers. It uses some training data to train itself and creates a fuzzy logic algorithm, which will establish a relationship between the input and the output. We have created a training data set and a test data set to train and test the neuro-fuzzy toolbox. The training data set includes three sets of inputs and one set of outputs. The inputs are 1. Normalized cost (execution energy), 2. Device energy, and 3. Latency. The output set is the decision. Unlike the fuzzy logic approach, we have not calculated the input data from the user inputs. Instead, we have followed our algorithm and the assumptions. For different input conditions and corresponding output conditions, we have generated the data sets. Using the inputs and outputs, the neuro-fuzzy designer establishes a system model, or a transfer function, and stores it as a fuzzy logic algorithm. After executing the training data, the neuro-fuzzy toolbox creates some variables and membership functions. The following figures represent those variables and membership functions.

Figure 9 represents normalized cost (execution energy) for the neuro-fuzzy analysis.

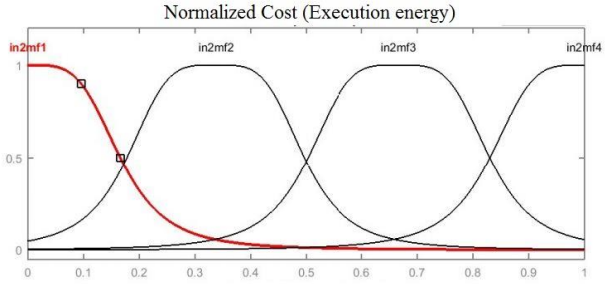

Figure 9: Normalized cost for neuro fuzzy

Figure 10 represents device energy for the neuro-fuzzy analysis.

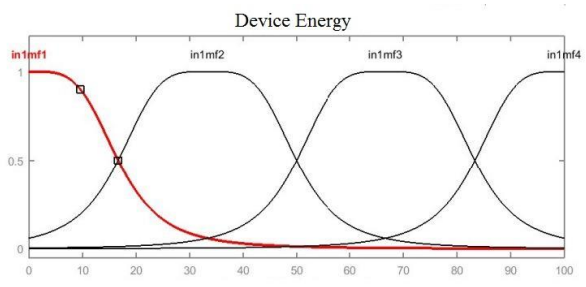

Figure 10: Device Energy for neuro fuzzy

Figure 11 represents latency for the neuro fuzzy analysis.

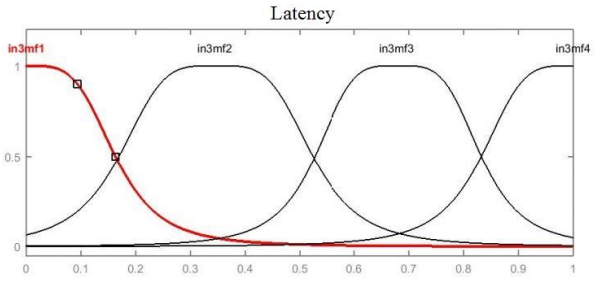

Figure 11: Latency

Figure 12 represents output decisions for the neuro-fuzzy analysis.

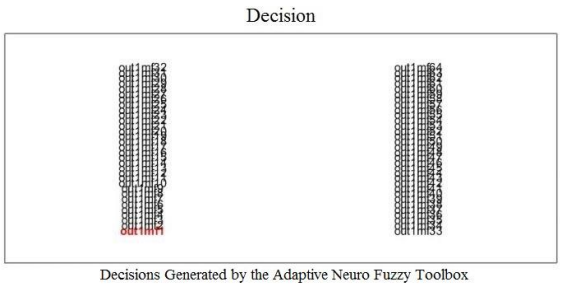

Figure 12: Decisions

After creating the fuzzy system using the adaptive neuro fuzzy designer toolbox, we have used it to implement our algorithm. Just like the fuzzy approach, we have used the new fuzzy system to decide whether to offload to the cloud or not based on different user inputs. We have used the testing data set to compare the neuro-fuzzy system with the fuzzy system. We have implemented both the fuzzy system and the neuro-fuzzy system separately for different input images and made decisions.

3. NUMERICAL ANALYSIS: AN EXAMPLE

A. Image Comparing Software

To better demonstrate our algorithm and how it works, we take a sample application- an image comparing software, which takes two images as input and compares percentage similarities between them. Our software uses corner detection to compare between two input images. We have used the "canny" edge detector algorithm to find the edges of the two images. Then we have compared the edges of the two images to find total matches. Finally, we have divided total no of matches by total no of data points to find percentage similarities between the two images.

As mentioned before, we assume that we have a software clone of our device running in the cloud. The clone has the same image matching software installed in it. We have introduced images of different sizes and types to test our algorithm. We have also provided different network conditions: different latencies and bandwidths, different device power levels to make a decision on application offloading. This decision-making is done by our algorithm. We have implemented both the fuzzy and the neuro-fuzzy system to make the decisions. The fuzzy system has user defined variables, membership functions and rules. From the inputs, it calculates the normalized cost (execution energy), device energy, and latency. Then it makes a decision based on its membership functions and rules. On the other hand, the neuro fuzzy system has trained itself and generated variables, membership functions and rules. Using its self generated variables, membership functions and rules, the neuro fuzzy system makes a decision. The two systems were implemented separately. We have used the same inputs for both systems. Both of them show similar outputs for the same inputs.

Table 2 shows the images that were used to test our algorithm:

Table 2: Properties of images that were used

#	File	Name	File resolution (pixel*pixel)	File size
1		flower.jpg	1920 * 1438	554 kB = 567296 B
2		flower 2.jpg	1920 * 1438	411 kB = 420864 B
3		flower 3.jpg	6000 * 4000	4.27 MB = 4477419 B
4		flower 4.png	1920 * 1438	6.20 MB = 6501171 B
5		flower 5.jpg	515 * 515	71.2 kB = 72908 B
6		flower 6.jpg	448 * 336	24.1 kB = 24678 B
7		flower 7.jpg	7 * 6	7.23 kB = 7403 B

Table 3 shows the output of the image comparing application for different input images:

Table 3: Output of image comparing app.

#	Input images	Similarity (%)
1	1 & 2	69.18
2	1 & 3	0.8346
3	3 & 3	100
4	1 & 4	95.8791
5	3 & 5	0.1093
6	2 & 6	0.3440
7	2 & 7	0
8	3 & 7	2.4*10^-04
9	1 & 7	0
10	2 & 5	0.9322
11	7 & 7	100
12	3 & 4	0.8246

Table 4 shows us the inputs, the outputs of the software and the decisions made by our algorithm.

Table 4: Inputs and output of the fuzzy algorithm

Input image	Cost	Latency difference	Device energy	Output decision
1, 2	0.1104	-0.1104	100	Device
1, 3	0.5352	-0.5352	100	Cloud
3, 3	0.9600	-0.9598	100	Cloud
1, 4	0.1104	-0.1104	10	Cloud
3, 5	0.4853	-0.4853	50	Cloud
2, 6	0.0582	-0.0582	5	Cloud
2, 7	0.0552	-0.0552	100	Device
3, 7	0.4800	-0.4800	100	Cloud
1, 7	0.0552	-0.0552	50	Cloud
2, 5	0.0605	-0.0605	100	Device
7, 7	1.62E-6	1.62E-6	100	Device
3, 4	0.5352	-0.5352	80	Cloud

Now, let's recall our algorithm and see if the output data matches them or not. For the first example, the comparison between image 1 and image 2, we see that the latency is close to zero. So, other parameters will come into consideration. As the cost of execution is low, and device energy is high, so our decision is execution is device.

For the third example, since the latency is negative as well as a very small value, the other parameters will not be considered. The decision will be to execute in cloud.

For the sixth example, since the latency is not a very small value, both the cost and device energy will be considered. Since the cost is low, and the device energy is also low, the decision will be execution in the cloud.

4. EVALUATION

We have simulated our algorithm in MATLAB®, tested it with sample software, and derived the outputs. We have implemented our algorithm using two different systems and generated outputs for both systems. We found out that both systems generate same outputs for the same inputs. We have compared our results to a previous approach. In [11], the authors implemented and tested their algorithm, CloneCloud, to offload computation to the cloud. The objective of CloneCloud is to partition the software into several threads/partitions and offload the threads to the cloud which will save time and energy. At first, they do the static analyzing to partition the software into different threads. Then, they do the dynamic profiling based on the execution time and energy consumed to select which thread/partition should be offloaded so that minimum execution time and energy can be achieved. So, their total execution time is as follows:

Clonecloud total execution time = Static analyzing time + Dynamic profiling time + Execution time of the software till the thread/threads which will be offloaded + pausing time + time to transmit the thread and necessary data to cloud + execution time in cloud + time to receive the output of the thread/s from cloud + resuming time + execution time for the rest of the software + time to print output

Whereas total execution time for our algorithm consists of the following:

Execution time of our algorithm= Dynamic decision making time about offloading based on user inputs, network characteristics, & device energy + time to transmit input data from device to cloud (if the decision is to offload) + execution time in the cloud + time to receive output data from cloud + time to print output

If we compare the two algorithms, we can see that if both decide to offload to the cloud, then our algorithm will take less time compared to their approach.

In [11], the authors implemented a virus scanning application in a mobile device, whereas we have implemented an image comparing software algorithm. We then performed the comparison based on input file sizes and observed the time required to offload for both the algorithms. Table 5 shows the comparison between the algorithms.

Table 5: Comparison

#	Input file size	Execution time in cloud(sec) (CloneCloud)	Execution time in cloud(sec) (our algorithm)
1	100kB	0.2	0.000208
2	1 MB	2.2	0.0028
3	10 MB	22.5	0.0267

We can see that our algorithm saves significant time offloading to the cloud compared to the approach in [11].

Also, for the mentioned offloads, our algorithm saves energy of the device. We have assumed that for each iteration, energy consumed in ARM processor is 10 µW, and calculated the amount of power saved. Figure 13 shows how much power is saved for each input file sizes.

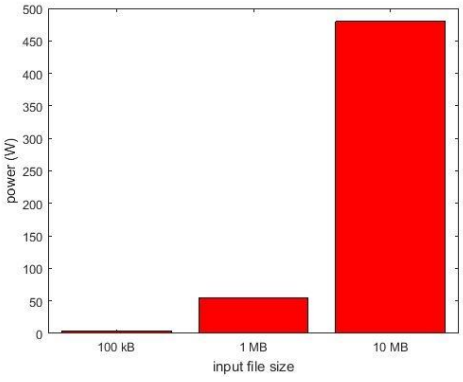

Figure 13: Device power saved by offloading to the cloud for different file sizes

We have also compared the two systems that we've implemented. We have used a testing data set to compare the two systems. We have used the neuro fuzzy designer toolbox in MATLAB® to generate error signal for the testing data set for both systems. Finally, we have plotted the two error signals so that we can compare them. We have plotted the error signals against epochs.

Figure 14 represents the error signal for the fuzzy system.

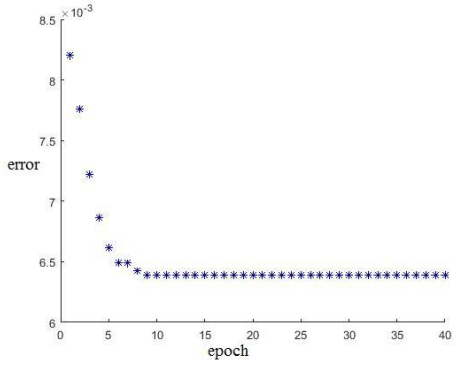

Figure 24: Error signal for fuzzy system

Figure 15 represents the error signal for the neuro fuzzy system.

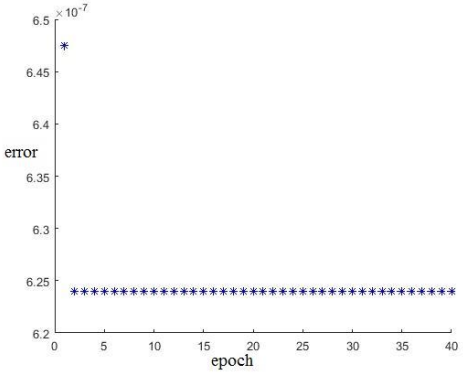

Figure 15: Error signal for neuro fuzzy system

From the two figures, we can see that the neuro-fuzzy system generates smaller error compared to the fuzzy system. Also, we can see that the neuro-fuzzy system converges faster than the fuzzy system.
From the two figures, we can conclude that the neuro-fuzzy system is better than the fuzzy system.

6. CONCLUSION

In this paper, we have focused on offloading computation to the cloud in a strategic way to ensure optimum utilization of mobile device resources. Our algorithm considers network conditions, mobile device energy, user inputs, and dynamically decides whether to offload the computation to the cloud or not. For better application in real life decision-making, fuzzy logic is used to implement our algorithm. We have simulated our algorithm in MATLAB®. We have implemented an image comparing application to evaluate our algorithm and gathered data for different inputs. Finally, we have compared our algorithm with previous approaches and found that our algorithm performs better than compared to a previous approach. And it also saves significant device energy by offloading to the cloud.

ACKNOWLEDGEMENT

We gratefully acknowledge the following:
(i) Time grants to access the facilities of the Open Cloud Institute of University of Texas at San Antonio.

REFERENCES

[1] https://www.apple.com/iphone/
[2] http://www.samsung.com/us/mobile/cell-phones/all-products?filter=galaxy-s
[3] http://www.microsoft.com/en/mobile/phones/lumia/
[4] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html
[5] http://science.time.com/2013/08/14/power-drain-the-digital-cloud-is-using-more-energy-than-you-think/
[6] S. Hao, D. Li, W. Halfond, R. Govindan, "Estimating mobile application energy consumption using program analysis", ICSE '13 Proceedings of the 2013 International Conference on Software Engineering, Pages 92-101, IEEE Press Piscataway, NJ, USA , 2013
[7] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X, Zhang, "ThinkAir: dynamic resource allocation and parallel execution in cloud for mobile code offloading", MCS '13 Proceeding of the fourth ACM workshop on Mobile cloud computing and services, Pages 9-16, ACM, New York, NY, USA, 2013
[8] A. Belogazov, J. Abawajy, R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing", Future Generation Computer Systems, Volume 28 Issue 5, May, 2012, Pages 755-768, Elsevier Science Publishers B. V. Amsterdam, The Netherlands
[9] G.P. Perrucci, F.H.P Fitzek , J. Widmer, "Survey on Energy Consumption Entities on the Smartphone Platform", Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd, Pages 1-6, Yokohama, Japan
[10] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, R. Buyya, "Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges", Communications Surveys & Tutorials, IEEE (Volume:16 , Issue: 1), Pages 337-368, 2013 IEEE Publications.
[11] B. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, "Clonecloud: elastic execution between mobile device and cloud", EuroSys '11 Proceedings of the sixth conference on Computer systems, Pages 301-304, ACM, New York, NY, USA
[12] http://www.dell.com/us/dfb/p/poweredge-2950/pd
[13] http://www.anandtech.com/show/4991/arms-cortex-a7-bringing-cheaper-dualcore-more-power-efficient-highend-devices
[14] http://www.evdoinfo.com/content/view/4818/64/
[15] http://www.cnet.com/news/4g-lte-showdown-how-fast-is-your-carrier/
[16] Paul Rad, Mohan Muppidi, Aldo S. Jaimes, Sos S. Agaian, and Mo Jamshidi "A Novel Image Encryption Method to Reduce Decryption Execution Time in Cloud", the 9th 2015 IEEE International Systems Conference, April 2015.

[17] Paul Rad, Van Lindberg, Jeff Prevost, Weining Zhang, and Mo Jamshidi "ZeroVM: Secure Distributed Processing for Big Data Analytics" , World Automation Congress (WAC) 2014
[18] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, "Elastic application container: A lightweight approach for cloud resource provisioning," in AINA. Los Alamitos, CA, USA: IEEE Computer Society, 2012
[19] https://www.docker.com/

S M Azharul Karim received his B.Sc. degree from Bangladesh University of Engineering and Technology in April 2012. He then joined Dhaka Electric Supply Company Limited (DESCO) in October 2012 and worked their till December 2014 as an assistant engineer. Currently, he is doing master's in Electrical Engineering in University of Texas at San Antonio. He has published 2 conference papers.

John Jeffrey Prevost received his first B.S. degree from Texas A&M in Economics in 1990. He received his second B.S. degree in Electrical Engineering from the University of Texas at San Antonio, where he graduated Magna Cum Laude in December 2009. In 2012 he received his M.S. degree in Electrical Engineering, also from the University of Texas at San Antonio along the way to earning his Ph.D. in Electrical Engineering in December, 2013. During his academic career he received two awards for Best Conference Paper, first at the World Automation Congress 2012 conference (Puerto Vallarta, Mexico) and again at the System of Systems Engineering 2013 conference (Maui, HI). He is currently an Assistant Research Professor in the department of Electrical and Computer Engineering at UTSA. In 2015, he co-founded the Open Cloud Institute where he also leads the Research and Education thrust area. Prior to his academic appointment, He served many roles as a technical leader in the computer hardware and software industry. He has served as Director of Product Development, Director of Information Systems and Chief Technical Officer. He is an active consultant in the areas of complex systems and maintains strong ties with industry leaders. He has served in the various capacities at professional conferences such as General Chair, Publications Chair, and the Tutorial and Organized Session Chair. His is a member of Tau Beta Pi, Phi Kappa Phi and Eta Kappa Nu Honor Societies, and has been a member of IEEE since 2006. He has published 9 conference and journal papers. His current research interests include energy aware cloud optimization, cloud controlled robotics, cloud based communications, and quantum cloud computing.

Paul Rad (High performance Cloud Group Chair at Cloud Advisory Council) received his first B.S. degree from Sharif University of Technology in Computer Engineering in 1994. He received his master in computer science from the University of Texas at San Antonio, where he graduated Magna Cum Laude in December 1999. Currently, he is the Chief Research Officer and Assistant Director of Open Cloud Institute (OCI) at the University of Texas, San Antonio, TX, USA. He has advised over 200 companies on virtualization and cloud computing and he has over 70 industry and academic technical publications. He holds 12 US patents on clustering, virtualization, cloud computing and big data analytics with over 200 citations by top 100 leading technology companies such as Microsoft, IBM, Cisco, Amazon Technologies, HP, and VMware. In 2014, he founded the Cloud and BigData Laboratory at the University of Texas, San Antonio, TX, USA as the first Open Compute Project (OCP) Certification Laboratory in the North America. In 2015, he co-founded the Open Cloud Institute where he also leads the industry and academic research cooperation at the University of Texas, San Antonio, TX, USA. He is currently involved in research on cloud computing architecture and cloud federation, big data analytics architecture, image processing & machine learning, hyper-scale computing, container clustering and micro-service application architectures for cloud native apps.