A Dynamic Framework to Support Offloading Under Resource Constrained Environment

Author

Asad ur Rehman

NUST201463906MSEECS60014F

Supervisor

Dr. Asad Waqar Malik

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Information Technology

DEPARTMENT OF COMPUTING

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

May, 2018

A Dynamic Framework to Support Offloading Under Resource Constrained Environment

Author

Asad ur Rehman

NUST201463906MSEECS60014F

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Information Technology

Thesis Supervisor:

Dr. Asad Waqar Malik

Thesis Supervisor's Signature:____

DEPARTMENT OF COMPUTING

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

May, 2018

Approval

It is to certified that contents of the thesis entitled "**A Dynamic Framework to Support Offloading under Resource Constrained Environment**" submitted by **Asad Ur Rehman** have been found satisfactory for requirement of the degree.

Advisor: **Dr. Asad Waqar Malik**
Signature: -------------------------

Date: ---------------------

GEC Member 1: **Dr. Anis ur Rahman**
Signature: ------------------------

Date: ---------------------

GEC Member 2: **Dr. Arsalan Ahmad**
Signature: ------------------------

Date: ----------------------

GEC Member 3: **Dr. Muhammad Muneeb Ullah**
Signature: ------------------------

Date: ---------------------

Declaration

I hereby certify that I have completed this thesis titled as *“A Dynamic framework to support Offloading under resource constrained environment”* entirely on the basis of my personal efforts under the sincere guidance of my supervisor Dr. Asad Waqar Malik. All of the sources used in this thesis have been cited and contents of this thesis have not been plagiarized. No portion of the work presented in this thesis has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

Signature of Student

Asad ur Rehman

NUST201463906MSEECS60014F

Acknowledgements

Firstly, I would like to pay my gratitude to Almighty Allah who gave me courage and patience to complete this project. I express my deepest appreciation to my supervisor Dr Asad Waqar Malik. I am heartily thankful to my GEC members Dr. Anis ur Rehman, Dr. Arsalan Ahmad and Dr. Muhammad Muneeb Ullah for their continuous and constructive feedback to develop an understanding of the project.

I'm thankful to my parents, family and friends who were always there with support and encouragement. My success became possible only due to prayers of my mother, whose love and affection has always been inspirational throughout my life.

Abstract

Mobile phones have become an inevitable part in human life. In spite of the rapid growth in mobile technology resource scarcity is still an issue for mobile devices, they lack in computational intensive tasks such as speech recognition, image processing and augmented reality etc. As a solution a pervasive computing technique "Cyber Foraging" can be applied, where mobile devices with limited resources can offload their computational tasks or heavy work to stronger surrogate machines in vicinity. Offloading is considered as an effective method to decrease source consumption of mobiles. It extends the mobile life and save battery. In cloud computing, Cloudlet is an emerging technique to scale up the computational abilities of mobile devices. It allows mobile devices to communicate with resource rich available servers in their local network and mobile devices can offload their computational tasks to these high performance servers and results can be send back to mobile in real time. Our study demonstrate that high-powered cloudlets support various delay sensitive applications that can require computation in real-time. Contrary to previous works and proposed offloading techniques, that only focus specific offloading issues. Our proposed architecture based on various perspectives like automated selection of cloudlet, based upon available resources, an intelligent time based decision engine to decide between local or cloudlet computation.

In this thesis we focus on the integration of mobile and cloudlet technologies to develop a time and energy-aware mobile computational task offloading paradigm. In the proposed solution we took android mobile as a client of cloudlet server. To analyze the performance of offloading we choose face recognition as offloading task, as real time face recognition requires abundant resources and energy. Our proposed framework dynamically identify the cloudlet server in mobile phone vicinity and get the information of resources available on servers. The framework includes a time based decision engine which can intelligently decide between local and remote execution. In this way we, consider the dynamic changes and offloading conditions of the context, since offloading may not be beneficial in all the conditions. In order to benchmark our proposed framework, a face recognition application is implemented locally at mobile node. Moreover, similar application has been accessed through application-as-a-Service module implemented in the proposed framework.

We’ve conducted different experiments and analysis which shows that by offloading the intensive computations to the cloudlet we can save a significant amount of energy and mobile resources.

Table of Contents

List of Tables

Table of Figures

CHAPTER 1: INTRODUCTION

The emerging research in mobile cloud computing has extended the mobile life and efficiency with techniques such as mobile task or code offloading, where mobile devices can offload their data or computation to a high performance server within their network and results can be send back to mobile in real time .Task offloading is an effective method, as it saves mobile resources and reduce battery consumption. Cloudlets scale up the computational abilities of mobile devices. In this study, we proposed a dynamic offloading framework based on face recognition using mobile application and Linux based server module. Research carried out in this area shows that resource-rich cloudlets are helpful to reduce overall processing time when computational tasks such as face recognition done on powerful backend servers [1].

In recent years, mobile applications started to be abundant in different categories such as social network, gaming, health, travel, entertainment and business etc. The most desirable features of mobile devices are small size, light weight, long battery life and comfort [2]. "Mobility" is considered as one of the characteristics of mobile phones where users can continue their work in spite of their movement. However, mobility inherit some problems also such as lack of resources, limited battery and low processing speed. Secondary concerns are system characteristics, such as processing speed, storage capacity and memory power. Whatever the progress of mobile technology, these devices still have a lot of limitations. Computational mobile applications like augmented reality, speech recognition, image and video processing, optical character recognition, language translator, health detection and analysis do not achieve their goal up to the full potential. Since they are too computational and require a subsequent amount of energy and resources. Mobile cloud computing and cloudlets are considered as the solution to the limit of mobile devices where intensive processing tasks are offloaded to the nearby cloudlet or cloud and the results can be sent back to the mobile in real time [3].

Real-time Face Recognition application is such a type of mobile application that require high responsiveness, archiving resources and intensive processing. Among the most propitious biometric techniques, facial recognition is the most popular and is widely used in criminal investigations and by law enforcement agencies. Human faces have complex, significant and multidimensional visualization and a face recognition computational model development is a complex procedure [4]. Many computational models have been proposed for effective facial recognition [1] [3] [5] [6]. When we have to develop such type of mobile applications, resource scarcity is a major problem. Mobile battery life can be extended by offloading intensive execution on nearby server.

To deal with WAN latency and mobile resource scarcity issues in 2009 Satyanarayan proposed a solution, known as cloudlet [7] as new architectural element in 3-levels hierarchy: mobile-cloudlet-cloud [8]. A cloudlet can support multiple virtual machines, it can help to reduce hardware dependency. Mobile devices can offload their computational tasks on these VMs and thus save their resources.

1.1 Motivation

Resource scarcity is a major problem in mobile devices. Mobile devices lack in computational intensive tasks and results in higher energy consumption and battery drainage. To deal with these problems, cloudlets have been introduced as a middle layer between cloud and mobile device. Mobile devices can offload their intensive tasks on cloudlet and can reserve their resources. Cloud computing is providing a cost effective and efficient data management services globally and this trend is gradually increasing in Pakistan. Organizations are looking for cloud-based platform to deploy applications, so that they can eliminate the cost and save energy. Mobile cloud has been widely considered to increase responsiveness and saving energy. The architectures which can offload the intensive mobile tasks to the cloud will play a pivot role in future to enhance the lifetime and efficiency of mobile devices. Global smartphone utilization has been increased drastically in recent years and it is estimated that one third of the worldwide consumers will be using smartphones by the end of 2018. This is due to the availability of cheap smartphones available across the market, which increase the requirement of applications that uses fewer resources to accommodate low-end devices.

Another reason is that smartphones have limited battery size and capacity so it is extremely important to manage energy consumption optimally. Mobile phone code offloading is an approach that could encourage better energy savings for smartphone. Smart phone applications proliferation is on the rise, especially video/image processing, augmented reality and mobile gaming, which have similar characteristics to those of PC. In image processing, face recognition by using nearby clouds/cloudlets on the trend.

1.2 Problem Statement

Cellular devices such as smartphones, notebooks and smartwatches are restricted in terms of their computational capacity, battery limit and storage amount. These deficiencies avert mobile devices to perform computational intensive tasks. These problems are not just an interim limitations of mobile technology, but intrinsic to mobility.

Task execution and response time is one of the primary concern of the mobile users. Research indicates that mobile devices with low hardware profile lag in running computation intensive and content rich applications as they require abundant of computational capabilities. Usually mobile devices have to operate in real time so response time is important. But due to computational limitations of the mobile devices it can take a long time to get back the results.

Battery life is another standout feature required by the mobile users. As per research 75% of the mobile users want to get improve battery life of their mobile phones. Now a days smart phones are not only used for communication purpose but also for net surfing, social media and watching videos which seriously shorten the mobile battery life.

To deal with mobile resource scarcity and computational limitations, cloud computing is emerging as a solution due to its widely available features such as quality of service (QoS), rich hardware, scalability, resource pooling and elasticity. Cloudlet computing used to augment the capabilities of poor resource mobile devices. Mobile users can offload their computational intensive applications and tasks to surrogate servers in vicinity. Offloading is a productive solution to overcome mobile devices constraints, since it relieved the mobile devices from intensive computation and provide potential benefits like performance improvement, battery saving and reliability.

In this thesis we intend to design and implement an architecture consist of mobile thin client which can dynamically offload the task to the nearby cloudlet server based host which can receive the off-loaded task from client and send back the result after compiling it.

1.3 Solution Statement

The main objectives of this study is to dynamically offload the face recognition task to nearby cloudlet, automated discovery of cloudlet, decision between local and server task execution, decrease energy and CPU/RAM consumption of mobile devices. On the basis of mobile task offloading we defined the following objectives for our architecture.

1.2.1 Cloudlet Discovery: In this study we propose a strategy to identify a cloudlet for computation offloading. The cloudlet discovery is divided into two steps

In first step, mobile application discover nearby cloudlet in its Wi-Fi range.

In second step, cloudlet is selected randomly. Once a connection is established between mobile and server, cloudlet server share its available resource information with the client.

1.2.2 Dynamic task offloading on the basis of execution time: From Mobile device perspective the task execution time is defined as the total time taken to perform the task on mobile and the RTT duration between offloading the task to the cloudlet and receiving the results back from it. Response time plays key role in intensive computational mobile applications. If the computation is intensive, it will take a long time to execute and will fail to meet user's need. Thus we proposed a time based decision engine which can dynamically decide between local or remote execution. We are taking decision to offload the task on server on the basis of last task execution time on mobile and cloudlet.

1.2.3 Reduce Mobile resource and energy consumption: The energy consumption of mobile while performing the task locally or during offloading is another primary concern that we considered. By evaluating and estimating the tradeoff between the energy consumption we tried to optimize the mobile energy consumption using android profiler.

1.4 Structure of Thesis

The thesis report is organized as below:

- Chapter1 contains the detailed introduction to the domain, its scope and motivation behind the study.
- Chapter2 includes the literature review, where several mobile to cloudlet offloading techniques have been compared and critically analyzed to find the optimal solution.
- Chapter3 contains the information about system architecture, the technique and strategy used for the offloading task.
- Chapter4 contains the details about the implementation of the whole system, libraries and API’s used for the development on mobile and server side.
- Chapter5 contains the information about application, system interface and the results.
- Chapter6 includes the Future work and Conclusion.

CHAPTER 2: Literature Review

2.1 Introduction

Mobile Cloud Computing (MCC) provide mobile devices diversity of cloud services at lowest cost. But still these devices create problems in executing media rich and computer vision tasks because many mobile's application like healthcare sensors, augmented reality and big data analytics do not work with full potential as they are too computationally intensive. Furthermore the other applications like image processing, speech recognition, language translators, optical character recognition, online games, video processing, and wearable devices need strong computational resources. These deficiencies of the mobile phones are relieved by the usage of different augmentation techniques like energy augmentation, storage augmentation and application handling augmentation. Mobile Cloudlet has been applied to leverage the competencies of resource limited mobile devices. Cloudlet is a term which was first introduced by Satyanarayanan to explain the procedure of expanding computational tasks to surrogate servers in the close vicinity. It is also known as follow me cloud [9] and Mobile Micro cloud [10]. Different numbers of cyber-foraging systems have been established, that use multiple methods to leverage distant resources such as where to offload, what and when to offload etc. As the clouds are present at far flung places and in these cases offloading to that cloud will not looked to be perfect resolution since discontinuation and sway influenced the performance. Moreover when someone is beheading the computationally intensive mobile phone applications, the excellent achievement can be taken by vigorously partitioning the functions in moderate way among the mobile device and the clouds. The researchers are attempting to make a latest offloading arrangements which would be actively working with both energy and time optimization. Thus the researchers have proposed numerous architectures; those would be explained in this paper.

2.2 Different Cloudlet Architectures

There are many architectures for application or task offloading which outsource the computational tasks of the mobile phones at granularity level. Usually there are two main application or task partitioning approaches namely: static and dynamic partitioning. In static partitioning the computationally extensive code is defined or separated only once in the application. It can be done periodically and causally in dynamic partition. The compute intensive code or classes of mobile application are separated at start with the static application partitioning. The application partitioning can be done casually or periodically. In casual partition approach, a runtime profiler and solver partition the code, which works in necessary situations. Furthermore in the periodic partitioning a run time anticipation mechanism has been used to evaluate the use of computing resources on mobile phone device. The profiler check the need and fallibility of computing capabilities for the mobile phone application and if there would be any deficiency of the resources, the application will be segregated and the computational intensive components are offloaded towards a distant server at the runtime for further processing. After execution the results returned back to the application which is running on the phone.

2.2.1 Mobile Cloud Hybrid Architecture (Mocha)

Mocha architecture has been introduced by T. Soyata to support extensive-parallelizable mobile cloud operations [1]. It is an offloading framework which provides real time response such as object recognition in battlefield and face recognition applications. It works actively for accurate face recognition which efficiently covers all the image processing steps and specifically intensive computations. Furthermore the authors use special compute boxes with the capacity of massively parallel processing (MPP) to overcome the problems of latency. A mobile, cloudlet and cloud framework has been implemented in this solution and algorithms were established to diminish the overall response time. Moreover the smart phone devices and laptops can connect with the cloudlet using multiple network connections such as Bluetooth, Wifi and 3G/4G, on the other side it would be connected to the cloud. The mobile phone device is all accountable for taking the images and then sending those images to the cloudlet for further preprocessing. And if the cloud is present quit near and latency is low then phone device can directly send the image to the cloud. But as sending image to the cloud is very intensive network task and due to that some processing works e.g. feature extraction must be done on the cloudlet or at the device itself. Then after preprocessing of the data, it is offloaded to the cloud for further processing and at the end the result sent back to the mobile device.

In addition in this architecture mobile phone devices transmit data to the cloudlet where it would be stored and then the cloudlet server renews the status of network latencies and keep the variation in latency to reach out different cloud servers. So to decrease the communication cost a dynamic partitioning task is performed by the cloudlet to select the best server, which helps to maximize the quality of service (QoS).

2.2.2 VM Based Cloudlets

As a solution [7] to Cyber Foraging Cloudlet term was first introduced by M. Satyanarayan as a middle layer in a 3-layered hierarchy such as mobile device – cloudlet – cloud [8]. According to him cloudlet is a decentralized and self-managed source of rich computers with limited number of consumers at a time. And the cloudlet are multicore computers which are widely dispersed on designated places internally connected by the high speed Wi-Fi located near the mobile devices and these cloudlet servers are connected to Cloud through high speed internet. The mobile phone devices use Wi-Fi rather than using 3G or LTE because the Wi-Fi provides high speed connection with low power consumption [11]. By the use of the high bandwidth, low latency and single hop connection to the cloudlet, mobiles can get a real-time and highly interactive response. While processing the mobile application discovers the nearby cloudlet server and then offloads the computation-intensive code to it. Moreover if mobile user moves away from the cloudlet server, it can go offline or might be its efficiency get effected by connecting to a distant cloud server. These cloudlets can have their own small data centers and are widely dispersed on assigned areas or places such as in hospital, offices and shops, they are self-managed due to their own network system and have decentralized ownership.

As the cloudlet can also contain cached data that can be access somewhere else, so that the loss of a cloudlet would not be calamitous. These cloudlets have very effective approach of distributing multiple users even in an isolated environment. Every user is allotted a separate VM specific to his application on the cloudlet.

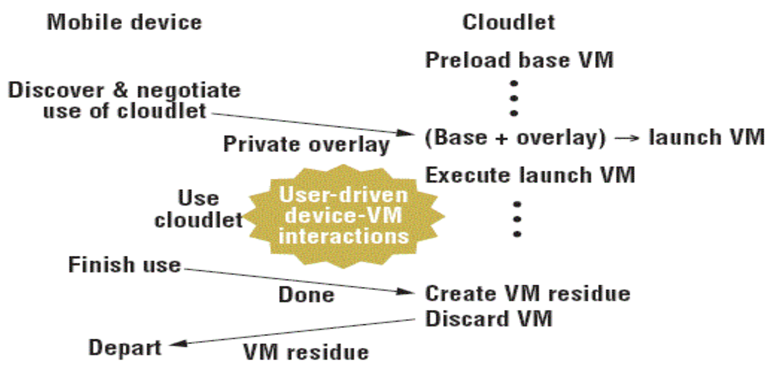


Figure 1 - Dynamic VM Synthesis

Dynamic VM Synthesis and VM Migration are the basic architectures for VM Based Cloudlets. VM Migration architecture delays the running VM on mobile phone device then transfers it to the cloudlet and continues its execution there. Especially the memory image of the source server VM is transmitted to the destination server without stopping execution .A clone image of live and seamless migration is obtained by copying the memory pages of the VM without affecting the operating system and other applications. Since the code is pre-copied, no code exchange is done during offloading. As far as the mobile device is concerned, VM Migration is heavy and it is time taking.

As the performance of dynamic VM Synthesis relies merely on local resources therefore the absence of WAN do not affect the synthesis process. Dynamic VM Synthesis dynamically provide a custom VM on the cloudlet and the mobile phone device discovers cloudlet in its LAN network using one of its discovery protocols. Moreover Dynamic VM Synthesis architecture uses VM technology to instantiate software service. When a TCP connection between mobile and VM cloudlet is established, mobile device can offload VM overlay. Then this overlay can be applied to the base-VM and launch the VM for user application.

The first and most important benefit of VM based cloudlets is the latency reduction as compared to the cellular networks as all the computation done on the nearby. The user application can be accessed directly on the cloudlet. Offloading the computation to the cloudlet saves money and mobile phone device energy [12].The primary advantage of the cloudlet is that it works efficiently even if is disconnected from the internet.

As cloudlets are considered as the intermediate elements between mobile and cloud, the mobile phone device does not require a connection with the enterprise cloud during the offload operation. It only needs to be connected with the cloudlet which is quite closest to it and it can continue its offloading even if it is disconnected from the cloud.

2.2.3 Mobile Assistance Using Infrastructure (MAUI)

The motivation behind the use of MAUI [13] is to save mobile energy and battery. As we all know that while using 3G connections suffers from long latency and slow data processing as compared to Wi-Fi and it also consumes much energy. So when the mobile code is offloaded to the cloud, the round-trip time (RTT) to cloud can be tens of milliseconds. Instead of this, if a nearby located server with a Wi-Fi is used, the RTTs will be less than 10ms.

Since offloading computation to cloudlet is considered as the best solution for the energy requirements of mobile phone devices, so many efforts have been made in this direction. It can be divided into two types. In first type the programmer specify how to divide a program and which sub parts should be executed remotely. The second type uses complete process or full VM migration. In this category, as the entire code and program sent to the remote cloud, the burden on programmers is reduced.

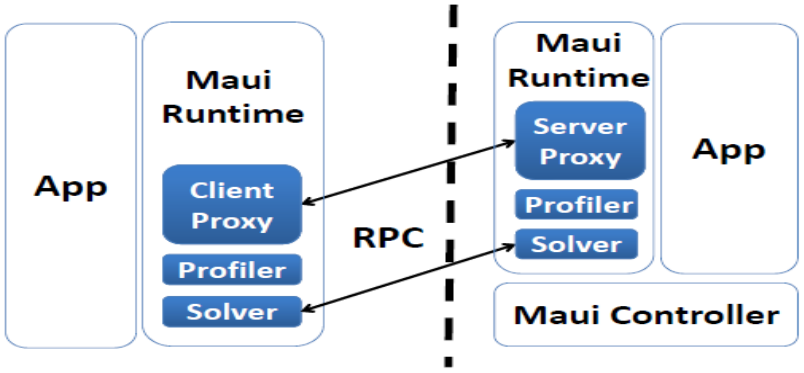


Figure 2 - MAUI Architecture

Mobile assistance using infrastructure is a complete system that achieves the advantages of the both types of remote execution which are explained above. MAUI facilitates fine-grained energy-aware code offload and remote execution using the .Net framework which redeems energy while minimizing the modification in applications. It can be achieved by utilizing the portability code, reflection, serialization and by maintaining code environment. Code portability is used to develop a smart phone application which can run locally or remotely by neglecting the difference in local and remote architecture. The programming reflection method used to identify the cost function of remotely executed methods. Serialization is accounted to resolve the size and cost of each method. This infrastructure provide a programming environment in .NET language and developers used annotations to decide between local and remote execution. MAUI optimizer determine whether it is beneficial to offload or not. Once the offloaded method execution is completed the framework collects the profiling information to figure out whether these methods should be offloaded further or not I future. If mobile phone is disconnected from the server the MAUI continues running the method on the android phone. MAUI utilizes the managed code to lessen the load on programmers and it also increases the energy benefits of offloading codes. It also takes in account that at runtime which method would be executed to save more possible energy. The developers established a conclusion that MAUI enhances mobile life and save energy by offloading source intensive applications. Mobile application which are Latency-sensitive, such as game applications can work effectively using the offloading mechanism.

2.2.4 Clone Cloud

The basic goal of Clone Cloud [14] is to allow fine grained flexibility in program partitioning and making decisions on what to run and where. It also provide automatic code partitioning. Clone cloud offload the intensive code to the remote cloud without any modification in the mobile phone application.It has static and dynamic code profiler to partition the application. The main purpose of clone cloud is to optimize the execution time and reduce energy usage. It partition the application at thread level. Clone cloud migrate the complete mobile VM instance to the cloudlet therefore the execution cost is high and time consuming. As offloading the entire VM to the server involves transmission cost and it consumes a lot of mobile energy.

2.2.5 ThinkAir

ThinkAir [15] is a framework which aims to provide on demand resource provisioning, it offers parallelism which provide method level offloading. According to the authors, it has the ability to adapt to the dynamic environment, flexibility for developers, better performance due to cloud computing. This framework used annotation methods to decide the offloading task or code. Any method which required to be executed remotely uses the annotation: @Remote. Execution controller is the part of the framework that decides when to offload. It collects data about current environment and previous executions and on the basis of these parameters it take offloading decisions. There are different parameters on the basis of which offloading decision is made including execution time, energy and cost. The execution controller also deals with server communications. Another part of the framework is the client handler which is responsible for connections, execution of the code and returning the results.

2.2.6 Code Offloads by Migrating Execution Transparently

COMET [16] is built on Dalvik Virtual Machine, is system designed allow runtime unmodified multithreaded applications to use more than one machine. This is done by letting the threads to move across machines according to the workload. It provides distributed shared memory structure with minimum communication between the machines. If network fails for some reason, another machine resumes computation.There are two main problems with this approach. Firstly, COMET can send unwanted data for execution which is which waste of bandwidth and resources. Second problem is because of the type of computation that is required by the mobile phones

2.2.7 Dynamic Performance Optimization

Dynamic performance optimization framework [17] which uses mobile agent based partitioning which required minimal structural changes on cloud. Before installing this application on mobile, it is divided into two components. The first part has a set of agent based application partition that can be offloaded to the cloud and a number of other processes that are always executed on mobile phone. Programmer assistance is needed so that offloaded components are identified correctly. The authors tested two real applications and later on stated that the framework can perform as well as it promises.

2.2.8 Chroma

R. K. Balan introduced a framework also called Chroma [18], which is based on tactics and remote execution system. Tactics is a compact form that contains information about the application which is important for deciding partition and offloading decisions. One of the important things in Chroma is that all the applications are isolated from the operating system and resources. The writers have ensured it that Chroma can be related with runtime systems that actually make partitioning decisions.

2.2.9 MACS

D. Kovachev and R. Klamma introduced a middleware service called Mobile Augmentation Cloud service [19] through which mobile application can be offloaded and executed over the cloud. Main purpose of this framework was to offload applications and ran on the cloud. The partitioning of the code for executing tasks on mobile and cloud is considered to be 95% energy saving and efficient in terms of computation power.

2.2.10 CUCKOO

Cuckoo [20] is another offloading model which consider the intermittent network parameters. It has very simple structure and mobile phone applications can easily benefitted from this offloading system. This framework provide a dynamic runtime profiler which make decision at runtime, that application will be executed locally or remotely. The main purpose of Cuckoo is to reduce energy consumption of the smartphones and to decrease the execution speed of intensive computations. Cuckoo is integrated with android framework and Eclipse development tool.

2.3 Evaluation

Better service, resource availability, improvement in security due to dynamic partitioning, reducing the latency are some of the important benefits of using MOCHA architecture. The problems are the speed, its complex architecture, continuous maintenance of latency tables and that it cannot be used offline. VM based cloudlet architecture is far better in terms of speed, simpler and improved QoS. Similarly it cannot be used offline and uses a lot of resources. VM migration is heavy and it is time consuming, decompression involve, pre-use customization and post cleanup overhead is involved. Customers have to rely on the service providers for the deployment of the cloudlet structure.

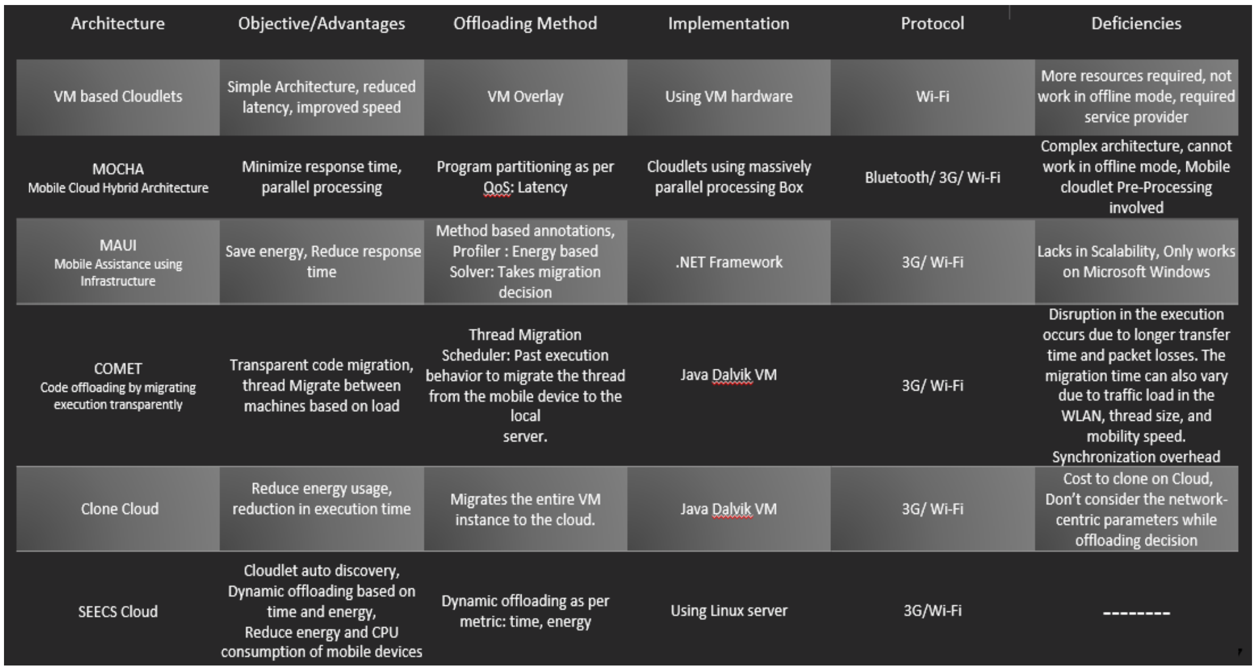
MAUI frame code is based on annotations and makes applications to reduce energy consumption and improve performance on mobiles. As per the tests, this method can only be used on Microsoft, with .Net created frameworks and thus it is not scalable.

Clone cloud efficiently divides the application by analyzing java code. It uses method level offloading and executes a cloned VM image which reduces the mobile CPU utilization. This method can be used to save power but the idea of pre-processing on mobiles can increase the cost and network traffic. Cost to clone complete VM image on Cloud is involved and it neglects network-centric parameters.

Think Air provides method level semi-automatic offloading. It is scalable and provide parallelism. Similarly COMET enables offloading of multithreaded applications using distributed shared memory.

In Chroma application must be partitioned manually and application offload tasks remotely using RPCs. The scheduler schedules the surrogates. MACS uses runtime partitioning technique and it keeps extra profiling information and resource monitoring.

This analysis covers different types of architectures for code offloading. Code offloading has been used to enhance overall performances of mobile devices. The good offloading technique is that decides the right time and the right thing to be offloaded.



Architecture	Objective/Advantages	Offloading Method	Implementation	Protocol	Deficiencies
VM based Cloudlets	Simple Architecture, reduced latency, improved speed	VM Overlay	Using VM hardware	Wi-Fi	More resources required, not work in offline mode, required service provider
MOCHA Mobile Cloud Hybrid Architecture	Minimize response time, parallel processing	Program partitioning as per QoS: Latency	Cloudlets using massively parallel processing Box	Bluetooth/ 3G/ Wi-Fi	Complex architecture, cannot work in offline mode, Mobile cloudlet Pre-Processing involved
MAUI Mobile Assistance using Infrastructure	Save energy, Reduce response time	Method based annotations, Profiler : Energy based Solver: Takes migration decision	.NET Framework	3G/ Wi-Fi	Lacks in Scalability, Only works on Microsoft Windows
COMET Code offloading by migrating execution transparently	Transparent code migration, thread Migrate between machines based on load	Thread Migration Scheduler: Past execution behavior to migrate the thread from the mobile device to the local server.	Java Dalvik VM	3G/ Wi-Fi	Disruption in the execution occurs due to longer transfer time and packet losses. The migration time can also vary due to traffic load in the WLAN, thread size, and mobility speed. Synchronization overhead
Clone Cloud	Reduce energy usage, reduction in execution time	Migrates the entire VM instance to the cloud.	Java Dalvik VM	3G/ Wi-Fi	Cost to clone on Cloud, Don't consider the network-centric parameters while offloading decision
SEECS Cloud	Cloudlet auto discovery, Dynamic offloading based on time and energy, Reduce energy and CPU consumption of mobile devices	Dynamic offloading as per metric: time, energy	Using Linux server	3G/Wi-Fi	--------

Table 1 - Comparison of Offloading Frameworks

CHAPTER 3: System Architecture

3.1 Introduction

This chapter presents a brief description of algorithm and technique used to implement the proposed architecture on mobile and server side. We highlight the proposed method used to compute the face recognition task and offloading to the cloudlet server

3.2 Computational Offloading Architectures

The idea behind computational offloading is known as cyber foraging [12], is to increase the storage and computing capabilities of mobile phones by taking advantage of opportunistic servers detected in the nearby environment [13]. Code offloading has been researched for over a decade now and had made many breakthroughs. Cloudlets are one of the most important task offloading solutions.

3.2.1 Cloudlets

Cloudlets are emerging rich decentralized internet architectures arising from the combination of mobile and cloud computing. These cloudlet servers offer resources and storage capacity to be used by the nearby mobile phones. They are considered as the middle layer of 3-layer hierarchy i.e. mobile, cloudlet and cloud. Their main purpose is to bring the mobile devices closer to the cloud, which means that the connection can be established via Wi-Fi instead of WAN. By doing this, connection delays can be reduced. The connection between cloudlet and mobile device can be considered as client-server relationship. In this case, mobile devices act as thin clients because the majority of data processing occurs on the server.

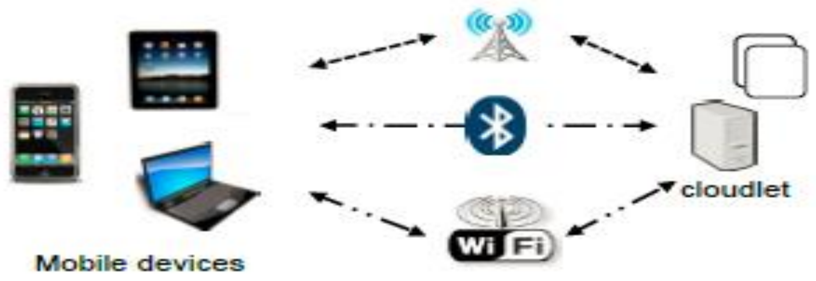


Figure 3 - Generic Cloudlet Architecture

3.3 Proposed Architecture

In recent years, number of cloudlet architectures have been implemented and proposed for computational offloading. As discussed in literature review section, there exist few cloudlet architectures for computational offloading. Our proposed framework is based on the two-layer mobile-cloudlet model. The first layer is mobile computation layer, whereas the second layer is cloudlet computational layer. The proposed framework aimed to empower the cellular users with a solution that help localized and custom dynamic computation offloading decision mechanism. Figure-4 describes a general architecture of the proposed solution. A cloudlet discovery service module used to find an appropriate cloudlet for offloading. A decision engine on mobile side gathers information such as execution time, battery level and processing speed. On the basis of these parameters the cost function module invokes offloading module and it decides between local processing and remote processing. In first scenario the processing is done on mobile device locally. If the processing is to be done remotely, the task is migrated to the available cloudlet via Wi-Fi. And after remote processing the results can be sent back to the mobile. A brief detail of components of proposed architecture in both layers are described as below.

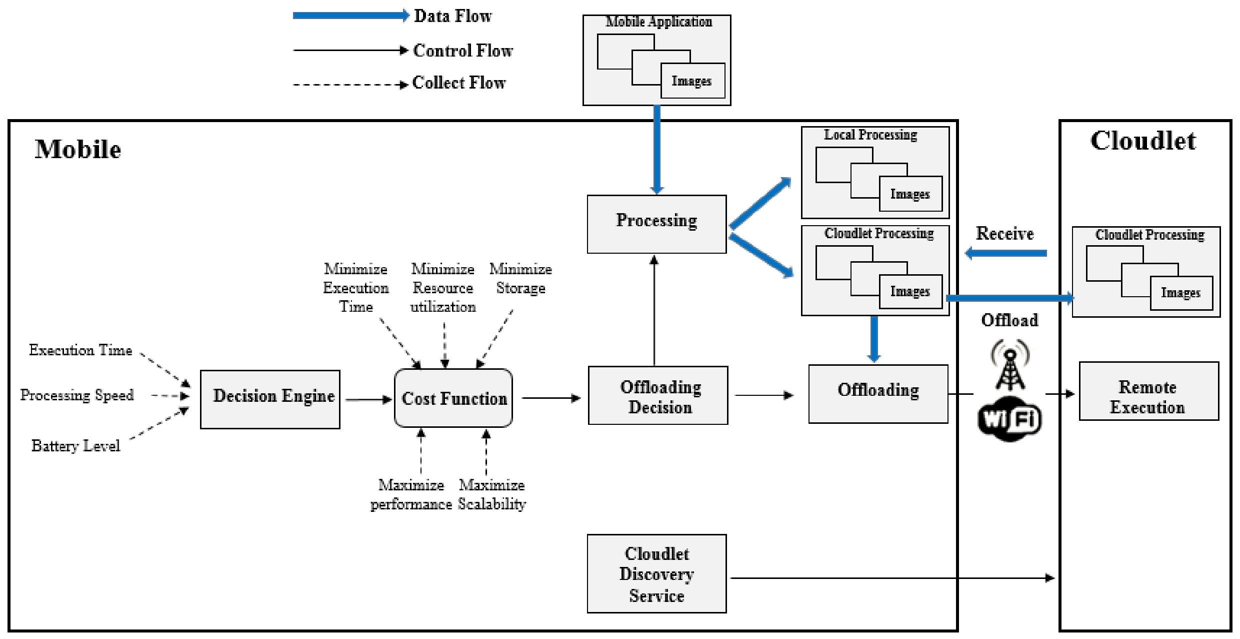
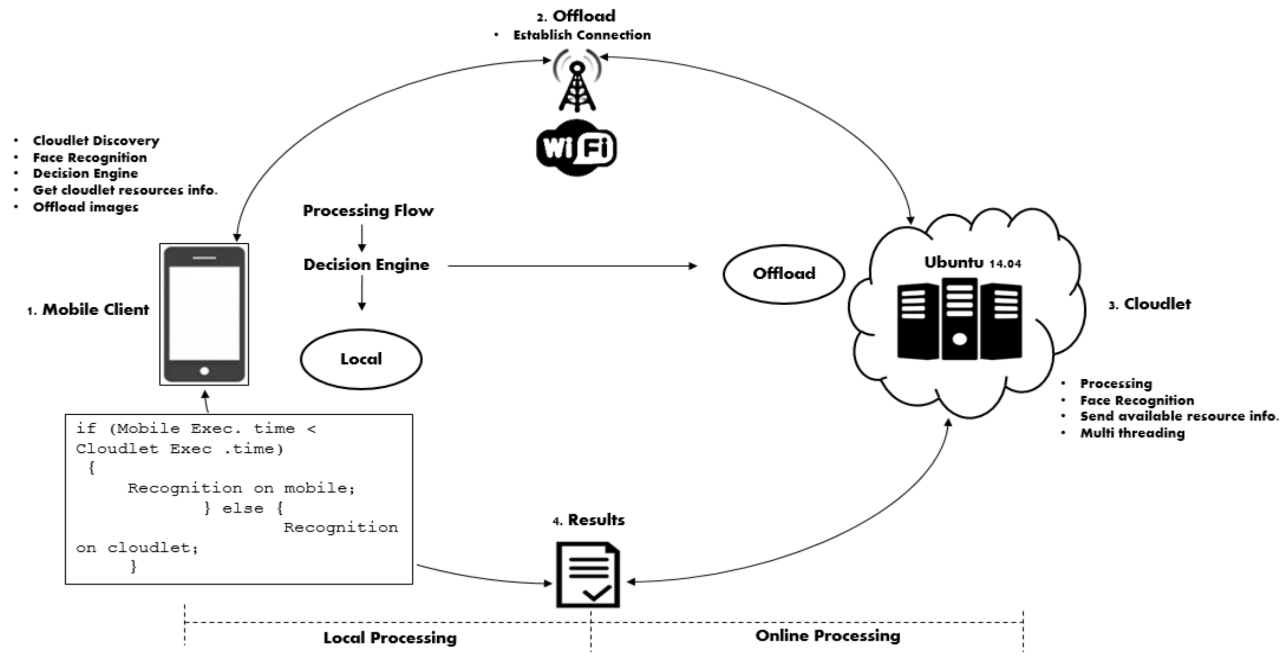


Figure 4 – SEECS Cloud System Architecture Diagram

Figure-5 shows high level process flow diagram for task offloading. A face recognition architecture is developed which is deployed on mobile device and cloudlet server. Mobile application can discover nearby cloudlet server, can get information about server resources and offload raw images to the Cloudlet. Mobile application can perform face recognition task locally and can offload it to the nearby server. When an offloading function is called, the offloading decision engine estimates the cost function in the mobile application and sends the raw images to cloudlet server. After all the processing on the server, results are sent back to the mobile.



```
if (Mobile Exec. time <
Cloudlet Exec .time)
 {
     Recognition on mobile;
             } else {
                     Recognition
on cloudlet;
      }
```

Figure 5 - High Level Process Flow Diagram

3.3.1 Mobile Node: Face Cloudlet

Layer 1 consist of Mobile application called “Face Cloudlet” which provides a user interface to interact the cloudlet server. The primary responsibility of this layer is to provide the face-recognition facility locally and a mechanism to interact with the cloudlet computing layer via Wi-Fi or 3g/4g services. A decision engine is implemented in the mobile architecture which intelligently decide whether the task will be computed on mobile node locally or it will be offloaded to the server. Mobile application capture images with camera and offload it to the cloudlet server for further processing. After processing results are sent back to the mobile.

3.3.2 Cloudlet Node: SEECS Cloud

Cloudlet layer contains the image processing and storage capabilities to detect and recognize the

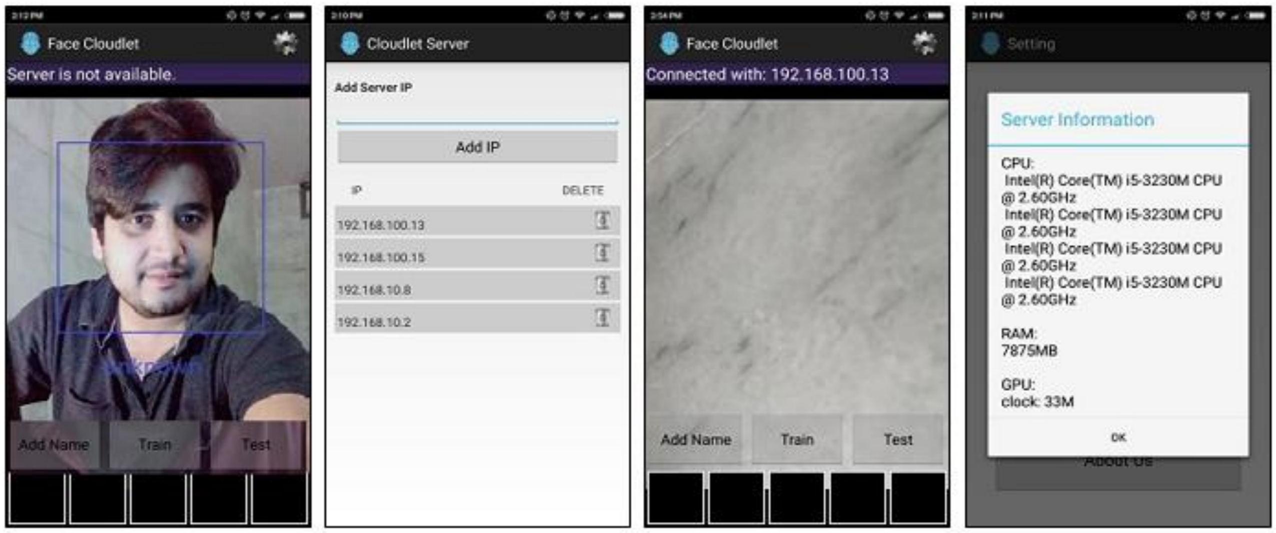
Human face and sent back the results to the mobile layer. In context of the data storage and computational processing, we provide a server based database to maintain all the media like images and metadata information received from the mobile node to avoid any additional computation and data storage overhead.

3.4 Face Recognition scenarios

In this work we implement two scenarios for face recognition ranging from local mobile computation to offload cloudlet. The below section provides a brief overview of the both scenarios.

3.4.1 First Scenario: Face Recognition on Mobile Node

In our study we took android mobile as a client of cloudlet server. We have developed a face recognition application "Face Cloudlet" using Android 7.0 Nougat. App uses mobile camera, storage and 3g-4g/Wi-Fi communication features and implement multi-threading. It has ability to recognize face locally or remotely by using the remote cloudlet server. In first approach, mobile phone can do all the processing itself. It can capture and process the images. After all the processing it save the training features in its local database. If the user feed an image to test, it can predict the user name by comparing the saved features and labels. Figure-6 show the main interface of the mobile application. The main interface screen has add name, train and test buttons. User will add his name and click train button, in response the application will take 5 images. After all the processing it will save the training feature in its local database.



Server is not available.

IP	DELETE
192.168.100.13	
192.168.100.15	
192.168.10.8	
192.168.10.2	

Connected with: 192.168.100.13

CPU:
Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz
Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz
Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz
Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz

RAM:
7875MB

GPU:
clock: 33M

Figure 6 - Mobile Application Face Cloudlet Interface

To deal with the storage issues we are not saving images in mobile locally. When user will click the test button, application will take an image and will predict the name of the person by comparing previously stored features. For Mobile application development we used a commercial face recognition library called Luxand. A brief introduction of the library is given below.

3.4.2 Luxand Face SDK

For local face recognition, we used a commercial library “Luxand FaceSDK” [21]. It provides a very good platform with supreme self-learning AI to recognize real-time face recognition within high quality live video transmission. The automated face tracking can be started on a simple function call as it does not require any predefined template or subject registration. The FaceSDK tracker API automatically authorize exclusive identifiers or sets specific names for different faces. Moreover it keeps on tracking the moving subjects and also receives scheduled alerts on the reappearance of those subjects. Its constant self-learning feature enhance recognition rates because it appears under different lighting angles and conditions.

The Luxand FaceSDK is a commercial face recognition library, it can be integrated into different types of client applications. It has an Application Programming Interface (API) that can not only detect and track different faces but also can note their facial features, age, and graphic expressions and smile. It has also the ability to recognize faces videos and images. It also has the ability to track and recognize the faces in the live videos. The API tracker makes it easy to work with video transmissions and provides features for tagging topics with their names and recognizing them even more.

The SDK face provides 70 characteristics of facial points e.g. the eye’s contours, eyebrows, mouth, nose and face etc. It has different processing cores which works together smoothly to speed up the recognition process. Furthermore this library also support webcams and IP cameras. It is a dynamic language library that is available for all types of operating systems like android, IOS and desktop systems.

3.4.3 Second Scenario: Face Recognition on Cloudlet

In this scenario, mobile application acts as a thin client. It capture images and send it to the cloudlet server for processing. After processing results will be sent back to the mobile. In this implementation mobile connection is established with the cloudlet server by using socket connection. User can set local server IP address in mobile application settings. Multiple server IP address registration is possible and app can scan all servers and select one of them for offloading. When app is opened, it discovers cloudlet server using IP address by broadcasting a test message to the existing server list. If server is existing, the app is connected to server and shows connected IP address. If it can't find server, it shows "Server is not available". When app finds cloudlet server, face-recognition module in server PC should be running by creating server socket. Once a server socket is created it is ready to accept the client request. The images are transferred to the server using this socket, when the connection between the mobile node and cloudlet server is successfully established. Cloudlet send its available resource information including CPU, available RAM and GPU to the mobile node. Figure-7 shows the high level offloading flow diagram. Socket connection are used for connectivity between mobile and cloudlet server. Once the images are received at server side, it detects the face in the image using facial landmarks. And then after cropping the images it extracts the image features and save in its local directory with labels. At server side we used an open source python library for face recognition, its details are in coming section.

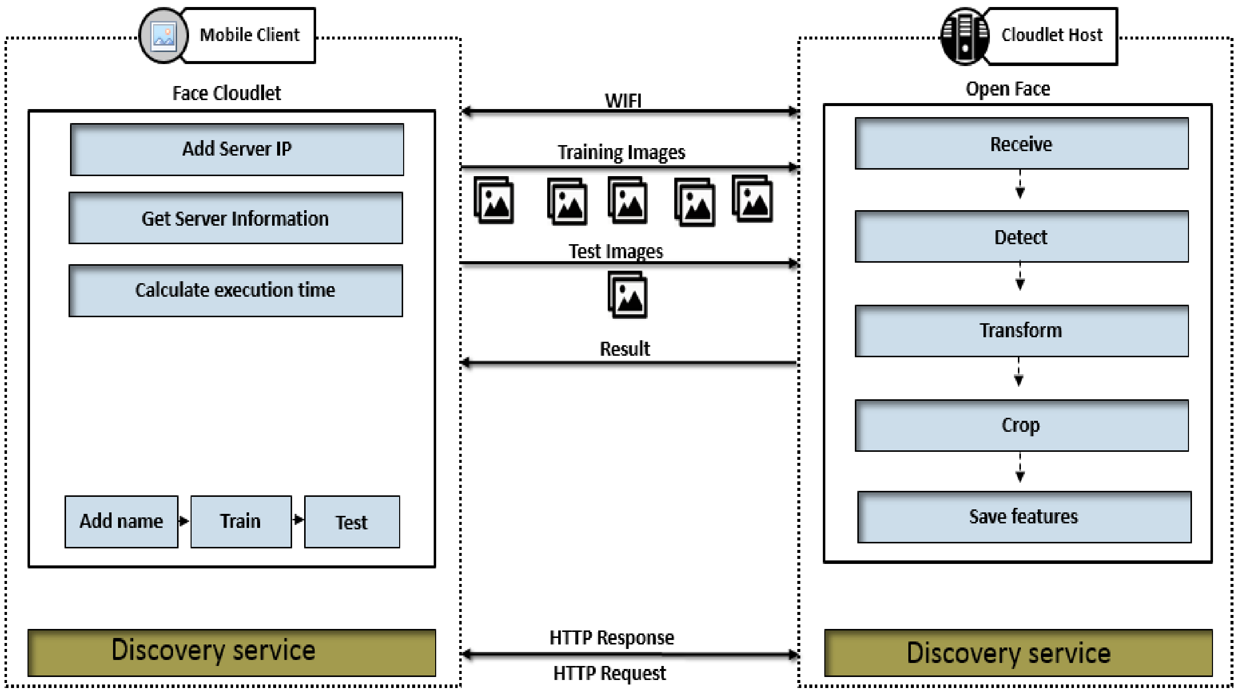


Figure 7 – Mobile to Cloudlet Offloading Flow Diagram

3.4.4 Decision engine

Face cloudlet includes a decision engine which take different parameters such as execution time, available energy and processing capacity as input. It can calculates the local and offloading execution time for training and testing images. It can intelligently decide between local and remote execution of task by comparing the execution time and other offloading parameters such as available battery and device CPU. Cloudlet selection and offloading decision making algorithm is very simple as below.

- ***Algorithm 1: Cloudlet Selection and making offloading decision***
- ***Input:*** *IP address,*T_{Mobile} *,* T_{Server}
- ***Output:*** *Cloudlet Selection and task offloading decision*
- *Let N is the number of cloudlets available near the cloudlet.*
- A is the set of applications that to offload the task to the cloudlets
  $A = \{A_1, A_2, A_3 \ldots\ldots\ldots\ldots A_N\}$ where
  $$A_i = \{T_{i1,} T_{i2,} T_{i3,} \ldots\ldots\ldots\ldots T_{ik}\}$$
  $$1 \le i \le N \textit{ and } K \textit{ is the number of tasks that belong to any application } A_P$$
- $User\ request\ to\ execute\ any\ task\ T_j\ that\ belong\ to\ application\ A_P$
- Execution time in offloading *the task* T_j *to cloudlet server is* T_{Server}
- Execution time for executing the task on mobile is T_{Mobile}

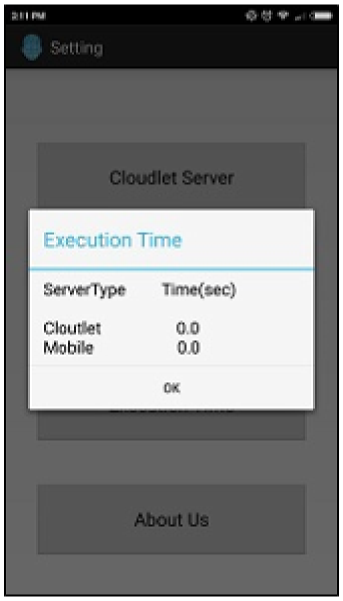
```
Procedure:
1:  Start
/*Selection of Cloudlet*/
2:  If multiple cloudlet IP's are available in the pool
3:    Mobile Application creates Client socket at its side and send connection request to the available cloudlet.
4:    Cloudlet share its resource information with the mobile. Connection established between the mobile and
cloudlet
5: Else if enter the possible Cloudlet server IP address
6: End If

7:  A user request to offload task $T_j$ , Application compare the previous execution time
Mobile and server
8:  If $T_{Server} > T_{Mobile}$
9:    Execute task $T_j$ locally at mobile device
10: Else if the task is offloaded to the cloudlet
11: End if
12: End
```

Figure 8 - Cloudlet Selection and offloading decision algorithm

If the time is less on mobile, face recognition will be done locally, otherwise it will use cloudlet server for the task. In this way we, consider the dynamic changes and offloading conditions of the context, since offloading may not be beneficial in all the conditions. Figure-8 show the application screen for execution time, at start the execution time is zero for both mobile and cloudlet.

Application keeps recent two readings for execution time and take offloading decision by comparing this time.



ServerType	Time(sec)
Cloutlet	0.0
Mobile	0.0

Figure 9 – Time based decision engine

3.4.5 OpenFace

Server module operates on Ubuntu 14.04. Framework is developed using Python programming language, using python face recognition SDK called Open Face and other python modules related to statistics and image processing. Open Face is designed to implement several classification algorithms and easy to use and update. Openface is a free and open-source face recognition library implemented in python and deep neural networks [22]

The openface workflow for face recognition process in a single image is given below [23].

1. In the first step it detect face using pre-defined model in opencv or dlib library.
2. It use real time pose estimation feature from dlib library and affine transformation feature from opencv.
3. Transform and crop the face for deep neural network.
4. The neural network represent the face on 128-dimension unit hyper sphere.
5. Apply any clustering or classification technique to predict the face recognition task.

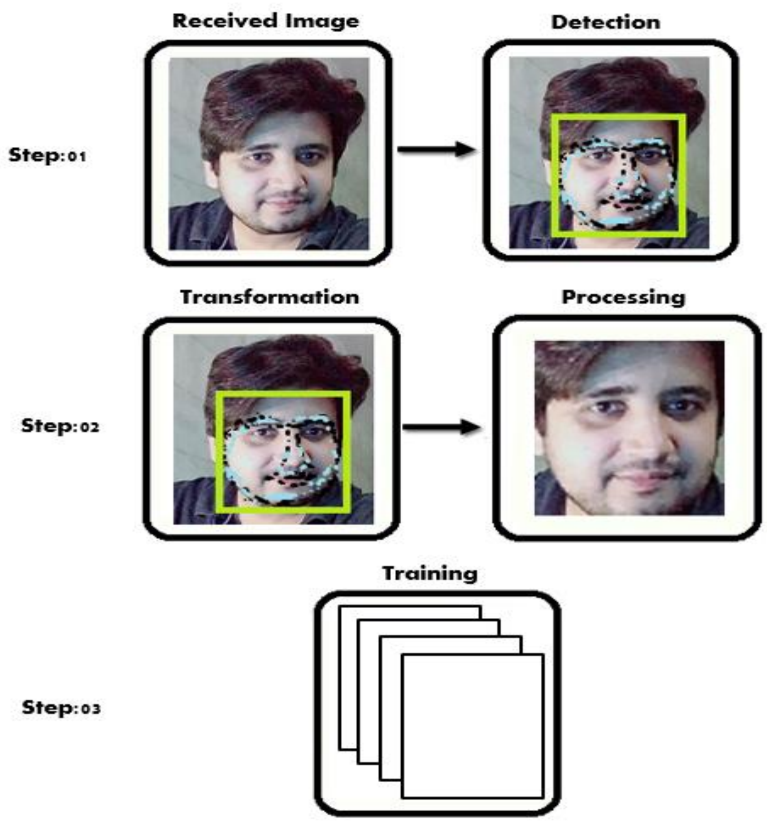


Figure 10 - Face Recognition process on cloudlet

3.5 How to start Openface Server

To run server program, follow the below steps.

1. Access Openface home directory
   asad@asad-HP-Pavilion-15-Notebook-PC:~$ cd openface-master/
2. Execute the "main.py" python script in home directory
   asad@asad-HP-Pavilion-15-Notebook-PC:~/openface-master$ python main.py
3. After script execution server will be started and create a server socket and ready to accept client request.
   - Socket created.
4. Once a socket is created successfully, it will be ready to accept client request
   - Socket Bind Success!
   - Socket is now listening
5. Now client will scan its local network and it will continuously send the message by the name of "test"
6. The message will be received at server end
   receiving name: test
7. Connection will be successfully created between client and server.

3.6 Training Data offloading

Once a connection is created successfully, user can send training and test images. For training process user will follow the below steps.

1. User will add his name in the first step
2. Name will be received at server end and it will create a directory in "facesamples" folder by user name "asad"
   ```
   receiving name:  asad
   ```
3. If the directory exist with the same name it will remove the directory
4. Now user will click "Train" button. FC will take 5 pictures and start sending it to server.
5. Images will be received in folder facesamples/asad
   ```
   receiving name:  asad
   37607./jpg
   ./facesamples/asad/1.jpg
   received, OK!
   1
   receiving name:  asad
   37608./jpg
   ./facesamples/asad/2.jpg
   received, OK!
   2
   === ./facesamples/asad/0.jpg ===
   === ./facesamples/asad/1.jpg ===
   === ./facesamples/asad/2.jpg ===
   === ./facesamples/asad/3.jpg ===
   === ./facesamples/asad/4.jpg ===
   ```

3.7 Training Data Pre-processing

Once all the images are received successfully in facesamples directory, server will call the pre-trained model for processing.

1. In the first step server call the "align-dlib.py" model available in the openface directory.
   ./util/align-dlib.py
2. Dlib model will align the images and after the face detection it will crop the image and saved into the facesample-aligned directory in 96*96 dimension
   ./facesamples-aligned/asad
3. If the aligned directory is already existing, it will remove the directory.
4. Metadata about all the images saved in the file "cache.t7" at path
   openface-master/facesamples-aligned/cache.t7

Figure-12 shows the images received at cloudlet server directory "Facesamples".After face detection program crop the face and save the cropped images in facesamples-aligned directory in 96*96 dimension.

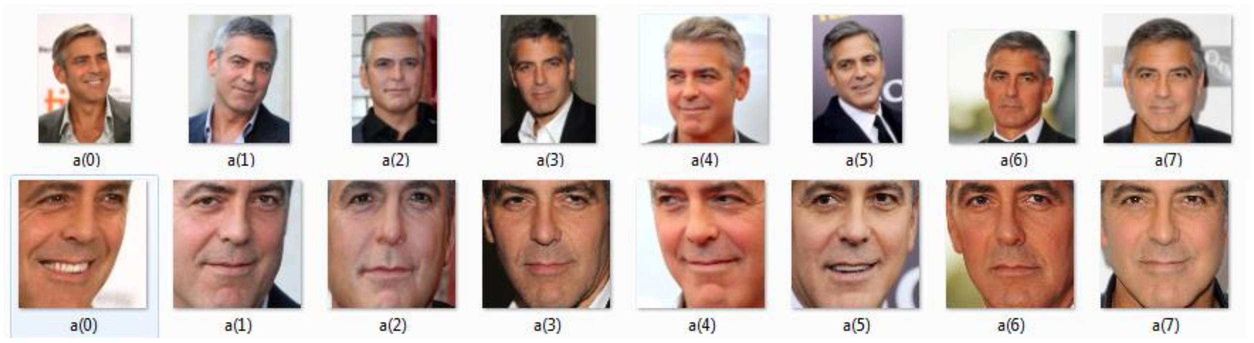


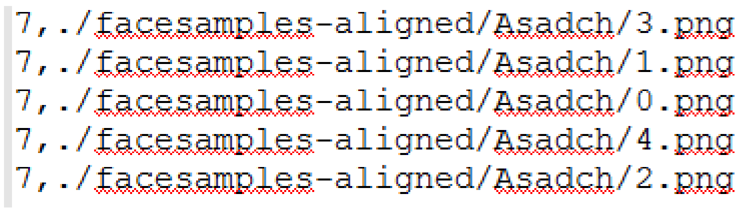
Figure 11 - Facesamples and Aligned faces

3.8 Running Training Model

After pre-processing is done Openface will call the training model

1. To start the training process call training model, we are using 'nn4.small2.v1.t7' torch model
   ./batch-represent/main.lua -- ./facesamples-aligned/asad
2. The model generate embeddings and save the transformation file as "reps.csv" at below path.
   openface-master/facesamples-featured/reps.csv
3. In the next step, a classifier is called to generate labels. We are using "Linear SVM" classifier.
   ./demos/classifier.py", "train"
   Classifier generate "label.csv" Figure-12 file and save it at below path
   openface-master/facesamples-featured/labels.csv
   label file contains 2 columns, one for labels and second for image path with name
4. Once training is finished model save the trained classifier at below path
   openface-master/facesamples-featured/classifier.pkl

The below figure shows the label.csv output file. Training model maintain index number, path and label of the input images.



```
7,./facesamples-aligned/Asadch/3.png
7,./facesamples-aligned/Asadch/1.png
7,./facesamples-aligned/Asadch/0.png
7,./facesamples-aligned/Asadch/4.png
7,./facesamples-aligned/Asadch/2.png
```

Figure 12 - Training Model Label File

CHAPTER 4: System Implementation

4.1 Introduction

This chapter provides a brief detail of system implementation. We used android as mobile implementation platform and Python programming language for cloudlet implementation. We used face-recognition as a workload to analyze the offloading performance. On mobile side we used an open source library Luxand Face SDK and on cloudlet side a face recognition SDK Openface is used. Further implementation details are described in the coming sections.

4.2 Openface Configuration

The first step involved in our implementation is the configurations required for OpenFace platform. We used Intel computer Core-i5 having Ubuntu 14.01 operating system. Server specification given as below.

COMPONENT	DESCRIPTION
SYSTEM	HP Pavilion 15 Notebook
CPU	Intel® Core™ i5-3230M CPU - 2.60GHz
HDD	750 GB
RAM	8 GB
OS	Ubuntu 14.04 64bit

Table 2 - Cloudlet server specification

4.2.1 System libraries and Build tools installation

We followed the below steps for the deployment of OpenFace dependencies

To install the dependent build tools and python libraries we followed the below steps. Open terminal and execute below commands

1. Update and upgrade the linux libraries
   sudo apt-get update upgrade
2. sudo apt-get install -y gcc g++ build-essential curl \
3. make git
4. Install python: sudo apt-get install -y python-numpy python-pip
5. sudo pip install

4.2.2 Opencv Installation

Opencv [24] is machine learning and computer vision library. It is an open-source library and support related application to run and accelerate. Opencv is prerequisite to make Openface work. In the first step we install opencv dependencies.

Install Opencv Dependencies

1. sudo apt-get -y autoremove python-opencv libopencv-dev
2. sudo apt-get -y install pkg-config libgtk2.0-dev libswscale-dev gfortran libavformat-dev unzip
3. sudo apt-get -y install python-numpy python-dev libjpeg-dev libpng-dev libjasper-dev libdc1394-22-dev libtbb-dev libtiff-dev

After installing all the dependencies, follow the below steps to configure opencv in home directory.

Install Opencv

To install the opencv open terminal and follow the below commands.

1. Make opencv directory in home
   mkdir -p opencv
2. Access directory: cd opencv
3. Command to get data archive: curl -L \
4. Url to Opencv: https://github.com/opencv/archive/3.1.0.zip \
5. -o opencv.3.1.0.zip && \
6. Unzip archive: unzip opencv.3.1.0.zip
7. Access unzipped directory: cd opencv-3.1.0
8. Make a new directory build inside: mkdir -p build
9. Access build directory: cd build
10. Set path variables
11. Set number of processes: make -j$(nproc)
12. Run install command to complete setup sudo make install

4.2.3 Dlib Installation

Dlib [25] is another open source machine learning library. Openface used it to get facial landmarks in real time face pose. It also provide GPU support on object detection. We've followed the below steps for dlib installation.

Dlib installation

1. Install dlib's dependencies
   sudo apt-get -y install cmake libboost-all-dev python-skimage
2. Create dlib folder in home directory
   mkdir -p dlib
3. Access directory: cd dlib
4. Get source code using URL: curl -L http://dlib.net/files/dlib-18.18.tar.bz2 \
5. -o dlib.tar.bz2
6. Unzip archive: tar -xf dlib.tar.bz2
7. Access examples directory: cd dlib-18.18/python_examples
8. Make build directory: mkdir -p build
9. Access Directory: cd build
10. Set AVX permission: cmake ../../tools/python –D USE_AVX_INSTRUCTIONS=ON 2>&1 | tee make.log
11. Build dlib: cmake --build . --config Release 2>&1 | tee build.log
12. Copy dlib source to python: sudo cp dlib.so /usr/local/lib/python2.7/dist-packages

4.2.4 Openface Installation

Open face is an open-source face recognition library implemented in python and shell programming languages.

1. Mkdir –p openface-master
2. Cd openface-master
3. Wget https://github.com/cmusatyalab/openface/archive/0.2.1.zip
4. Unzip 0.2.1.zip
5. Cd openface-0.2.1
6. Sudo python2 setup.py install

4.2.5 Important Openface Programming Classes

➢ **openface.AlignDlib class**

Dlib class used to detect face using pre-defined model with the help of facial landmarks. After alignment and preprocessing images used as input into a neural network model.Images are resized into 96x96 dimension.

➢ **openface.TorchNeuralNet class**

After face detection neural network training model run on cropped images. The neural network represent the face on 128-dimension unit. And save the extracted features in a file. For testing any face we can apply any clustering or classification technique to predict the face recognition task. Linear SVM is used as a classifier

➢ **openface.data module Class**

Data module classs used for data handling. It contains information like image metadata, class, label, name, path, load images from directory etc

➢ **openface.helper module Class**

It is a helper module which is used for debugging purpose.

4.2.6 Socket creation on Cloudlet side

1. Get Server IP
   HOST = "   '‘
   PORT = ‘’   ‘’
2. Create server Socket for incoming request
   s = socket.socket(socket.AF_INT, socket.SOCK_STREAM)
3. Bind the socket with a port
   s.bind((HOST, PORT))
4. Start listening for incoming TCP connections
   s.listen(v)
   conn, addr = s.accept()
5. Read Request from Client
   conn.recv(buffer)
6. Write Reply to client
   conn.sendall(msg)
7. Close connection
   s.close()

4.3 Mobile Application Implementation

Following steps are involved in the development of the mobile application.

1. Download FaceSDK Package and place it in project directory

2. Activate the package by using activation key, FSDK_ActivateLibrary used to activate the SDK library
3. Face_SDK initialization by using FSDK_Initlize function.
4. Load an image from file or buffer (FSDK_LoadImageFromFile, FSDK_LoadImageFromBfr)
5. Setting of face detection parameters to detect facial landmarks with the help of following functions (FSDK_SetFaceDtctnparamtr, FSDK_SetFaceDetectionThreshld).
6. To detect a single face (FSDK_DetctFace) function is used and for multiple faces (FSDK_DtctMultiplFaces) function is used in an image
7. Detect facial features in an image with the help of following function (FSDK_DtctFacialFeaturs, FSDK_DtctFacialFeatursInRgn)
8. To extract face template from image (FSDK_GetFaceTmplt) function is used and for extraction of face specific region (FSDK_GetFaceTmpltInRgn) function is used
9. To match different face templates and get the facial similarity level between the faces (FSDK_MatchFaces) function is used
10. To calculate matching threshold to detect if face belongs to the same person following functions used (FSDK_GetMtchngThrshldAtFAR and FSDK_GetMtchngThrshldAtFRR).

4.3.1 Socket Creation at mobile side

```
1.  Create Client Socket
2.  Connect to Host IP, Port=‘’   ‘’
3.  Create socket at server side object and bind port with it
4.  ServrSockt socServr = new ServrSockt(SERVR_PORT);
5.  Create socket reference of client at server side
6.  Sockt sokclient = null;
7.  Accept client connection and handover communication to server socket
8.  sokclient = sokServr.accept();
9.  Send Request to server
10. SendMessage sic = new SendMessage()
11. Read reply from server
12. BufferedReader br = new BufferedReader()
13. close
14. mySockt.close();
```

CHAPTER 5: Results and Evaluation

5.1 Introduction

This chapter provides a brief description of results and evaluation. There are 2 scenarios for face recognition which has been tested in this work i.e. face recognition module on mobile and face recognition module on cloudlet. We presented the data in tabular and graphical format to draw the logical solution. We calculated the face recognition time on mobile and cloudlet to compare where the best performance can be achieved. Subsequently we evaluate the performance of FACE-CLOUDLET by using two different image dimensions of 500*500 and 250*250. To evaluate the recognition time correlation with mobile device CPU, we performed face-recognition on two different mobile devices Redmi Note4 and Huawei TIT-AL00. Finally, we analyzed the CPU, battery and RAM consumption for the task execution in two cases i.e. Mobile recognition and Cloudlet recognition.

5.2 Experimental Setup

We used two different approaches for face-recognition. The face-recognition has been implemented on mobile and cloudlet. Our development platform is Android studio, Ubuntu 14.04, python libraries, opencv, openface and shell programming. There is only one cloudlet setup in our experiment and multiple clients can connect to it. The hardware details configured for the experimental setup are given below.

Component	Cloudlet Node	Mobile Node 1	Mobile Node 2
System	HP Pavilion 15 Notebook	Redmi Note4	Huawei TIT-AL00
CPU	Intel Core i5	Octa-core 2.0 GHz	4 Core 1.3 GHz
HDD/Memory	750 GB	32 GB	16 GB
RAM	8 GB	3 GB	2 GB
OS	Ubuntu 10.04 64 Bit	Android 7.0	Android 5.1

Table 3 - Experimental Setup

The Mobile client and cloudlet are connected through high speed broadband link 10/5 mb downstream/upstream with a single hop connection with round trip ping latency of max 4ms.

5.3 Results

The below section provides the detail implementation results of each module.

5.3.1 Face Recognition on Mobile Device (Redmi Note4 vs Huawei TIT-AL00)

Face recognition on mobile is the first benchmark module in the main functionality of this architecture. For performance analysis of both scenarios we performed the task using 2 mobiles of different specifications i.e. Redmi Note4 with 2.0 GHz CPU, 3 GB RAM and 13 megapixel camera vs Huawei TIT-AL00, 1.3 GHz CPU and 5 megapixel camera. Face recognition training and tests conducted 5 times for 5 different persons on each mobile device. Conducted experiments shows that results depend upon different factors like image resolution, device CPU and image size. The average training model time for face recognition was 11.06 sec on Huawei device and 3.62 sec on Redmi Note4. Similarly for face test model the average time was 1.98 sec on Huawei and 1.48 sec on Redmi Note4. Below tables represents the execution time for 5-training images and 1-testing image on mobile.

Execution Time in Seconds		
Training Model on Mobile	**Test Model on Mobile**	**Total Time**
11.35	1.82	13.17
11.26	1.89	13.15
10.84	1.9	12.74
11.03	1.84	12.87
10.81	2.46	13.27

Table 4 - Face Recognition on Mobile - Huawei TIT-AL00

Execution Time in Seconds		
Training Model on Mobile	**Test Model on Mobile**	**Mobile Total**
3.57	1.48	5.06
3.62	1.33	4.96
3.63	1.58	5.21
3.62	1.53	5.16
3.68	1.49	5.17

Table 5 - Face Recognition on Mobile - Redmi Note4

Table-4 and Table-5 clearly shows that processor speed effect the task execution time. Huawei mobile had less processing speed with 1.3 GHz CPU and results shows that total execution time on the device is almost 3times higher then Redmi Note4 which has 2.0 GHz CPU. The graph in

Figure-12 shows that the average execution total time for face regcogntion task on mobile is 13.04 seconds on Huawei and 5.10 seconds on Redmi Note4.

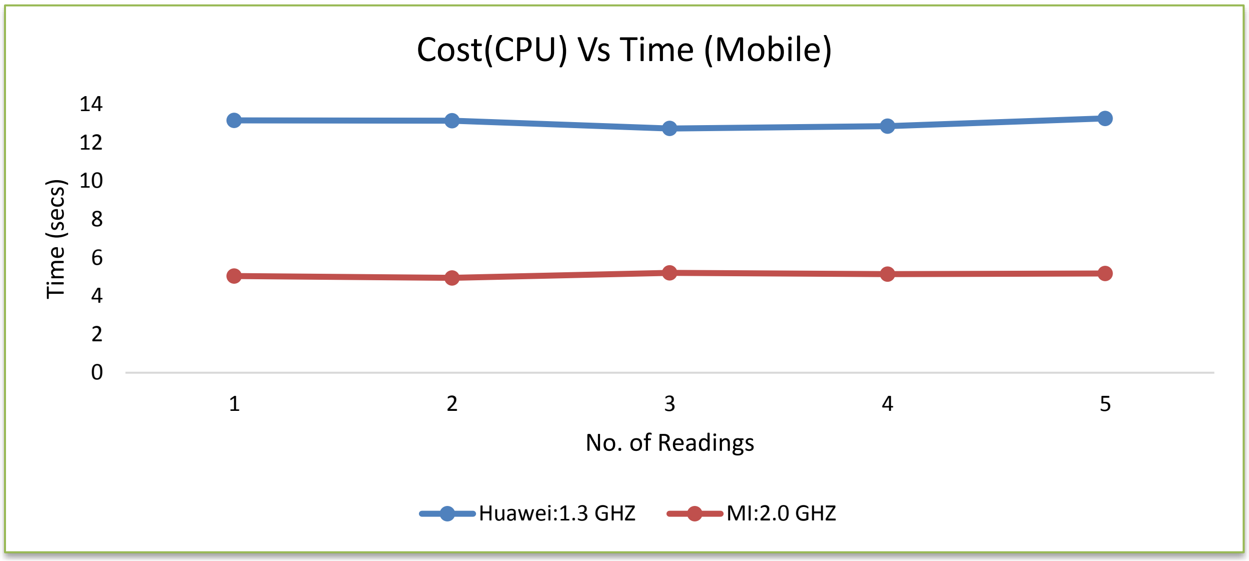


Figure 12 - Execution time for Face Recognition on Mobile Vs CPU

5.3.2 Face Recognition on Cloudlet (Redmi Note4 vs Huawei TIT-AL00)

Face recognition on Cloudlet is the 2nd benchmark module in the main functionality of this architecture. We deployed the Openface - A face recognition framework on our cloudlet server which has 3.0 GHz CPU and 8 GB RAM. The face recognition process conducted 5 times for different persons. Task offloading to cloudlet is done by using the same mobile devices i.e. Huawei and Redmi Note4. A decision engine is implemented in Face-Cloudlet to decide whether the task is worth to offload on cloudlet or not. To check the communication cost with cloudlet we calculated the time to send the training and test images to the cloudlet and time to get back the response. To deal with multiple client requests multithreading has been implement at server side. Task is offloaded to the cloudlet using a high-speed internet connection of 10mb connection. Results shows that client processing speed effect the face recognition execution time. Table-6 shows that the average execution time for training model on cloudlet is 24.34 and test model is 3.27 seconds using Huawei TIT-AL00. Table-7 shows that the average execution time for training model on cloudlet is 22.47 and test model is 2.63 seconds using Redmi Note4.

Execution Time in Seconds		
Training Model on Cloudlet	Test Model on Cloudlet	Total Time
24.67	3.31	27.98
23.08	3.35	26.43
24.66	3.6	28.26
23.22	3.08	26.3
26.12	3.03	29.15

Table 6 - Table 5-4 Face Recognition on Cloudlet - Huawei TIT-AL00

Execution Time in Seconds		
Training Model on Cloudlet	Test Model on Cloudlet	Mobile Total
22.36	2.7	25.06
22.77	1.87	24.64
22.1	2.8	24.9
23	2.95	25.95
22.12	2.85	24.97

Table 7 - Face Recognition on Cloudlet - Redmi Note4

Figure-13 shows that the average execution total time for face regcogntion task on cloudlet is 27.62 seconds on Huawei and 25.10 seconds on Redmi Note4.

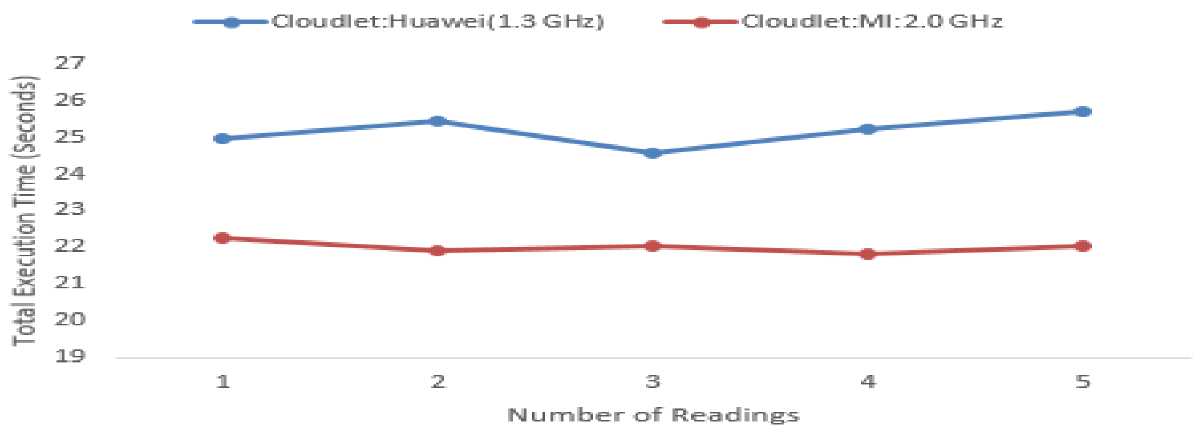


Figure 13 -Execution time for Face Recognition on Cloudlet Vs CPU

From conducted experiments it is evident that total execution time on mobile was less when we used the images with low resolution and size. As long as we keep increasing the image resolution

5.3.3 Face Recognition on Mobile Vs Cloudlet

It has been observed that the execution time on mobile was less as long as we were using images with low resolution. As we increased the image resolution the execution time on mobile is increasing. Figure-14 shows the comparison between mobile and cloudlet where we use an image with 500*500 resolution.

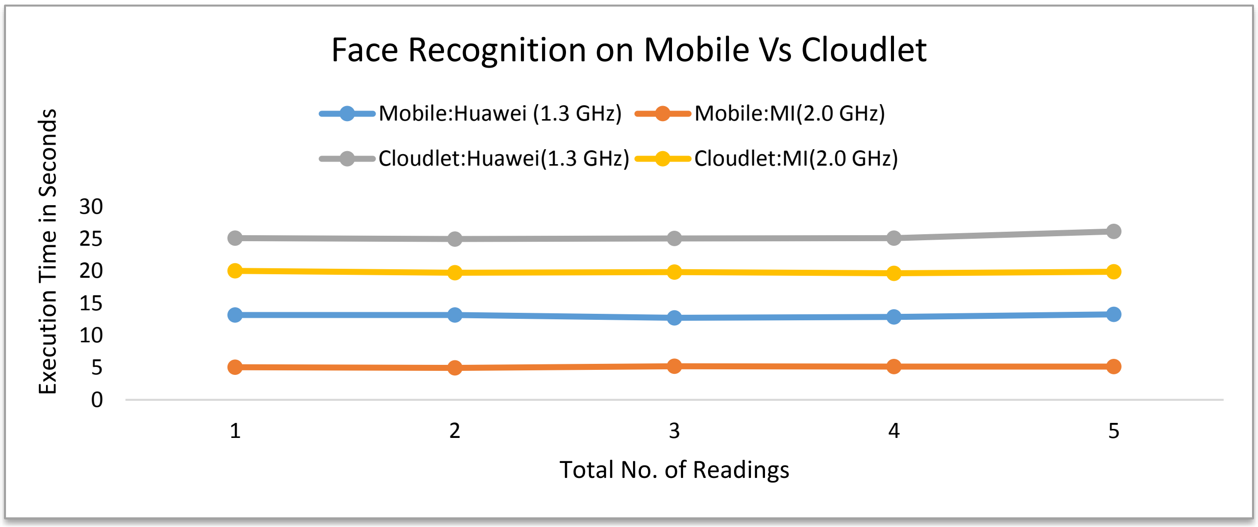


Figure 14 - Execution time for Face Recognition on Mobile Vs Cloudlet

We conducted different experiments with images of different resolution as shown in Figure-15. As the image resolution is increasing more then 800*800 mobile time is increasing then cloudlet time.

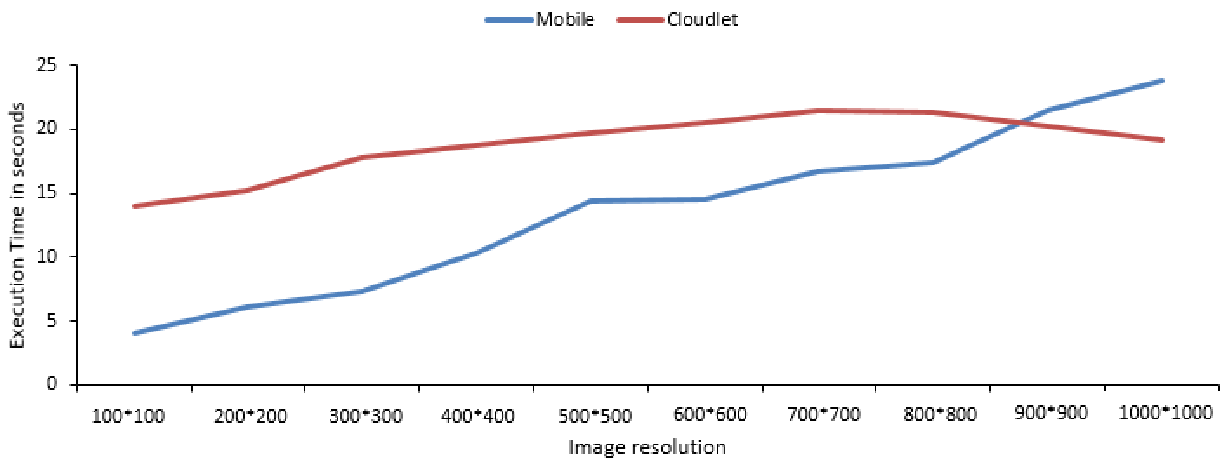


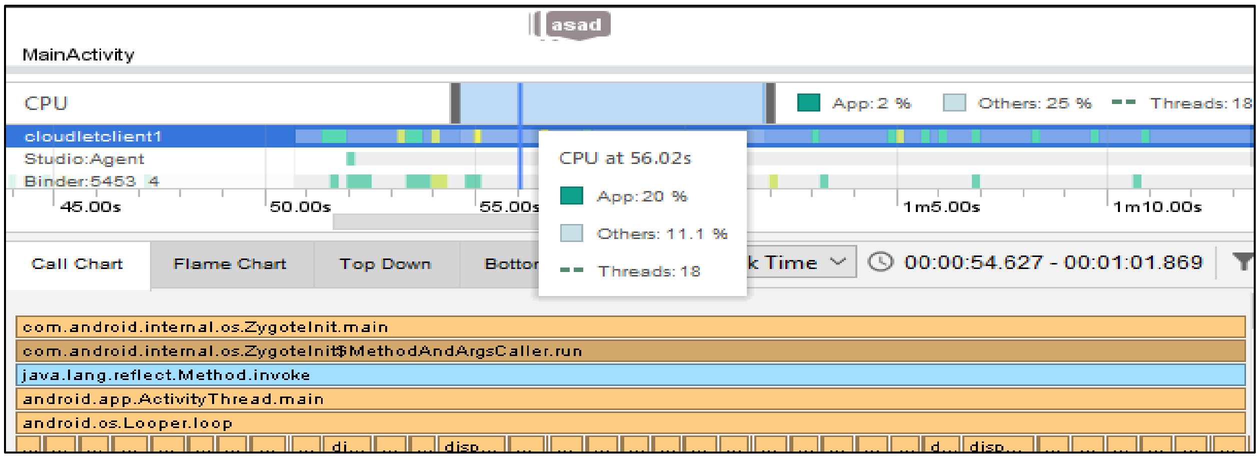
Figure 15 - Execution time vs data transfer size on Cloudlet

5.3.4 CPU Consumption

We analyzed the CPU and Memory of mobile application using android profiler which provide tools to monitor real time data and performance of mobile applications. There are following two cases.

- Consumption while executing task on mobile
- Consumption while executing task on cloudlet

The consumption of CPU while executing task on mobile has been shown in Figure-16. The maximum CPU consumption was 20%.



```
com.android.internal.os.ZygoteInit.main
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run
java.lang.reflect.Method.invoke
android.app.ActivityThread.main
android.os.Looper.loop
```

Figure 16 - CPU consumption on Mobile

The consumption of CPU while executing task on cloudlet has been shown in Figure-17. It is noteworthy that maximum CPU utilization was 17%. So it is concluded from results that it is always beneficial to offload the task to the cloudlet to save mobile resources.

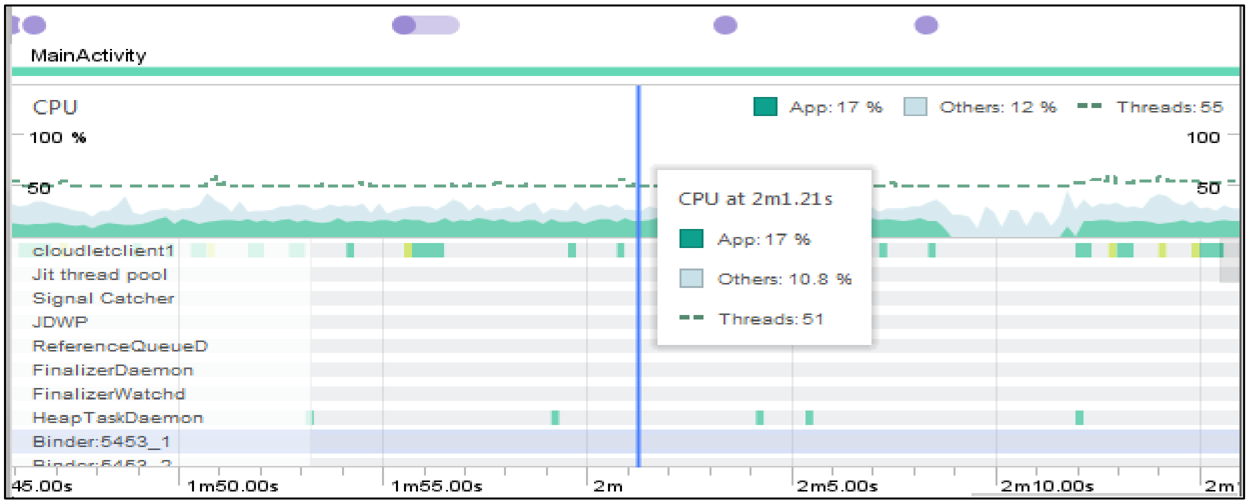
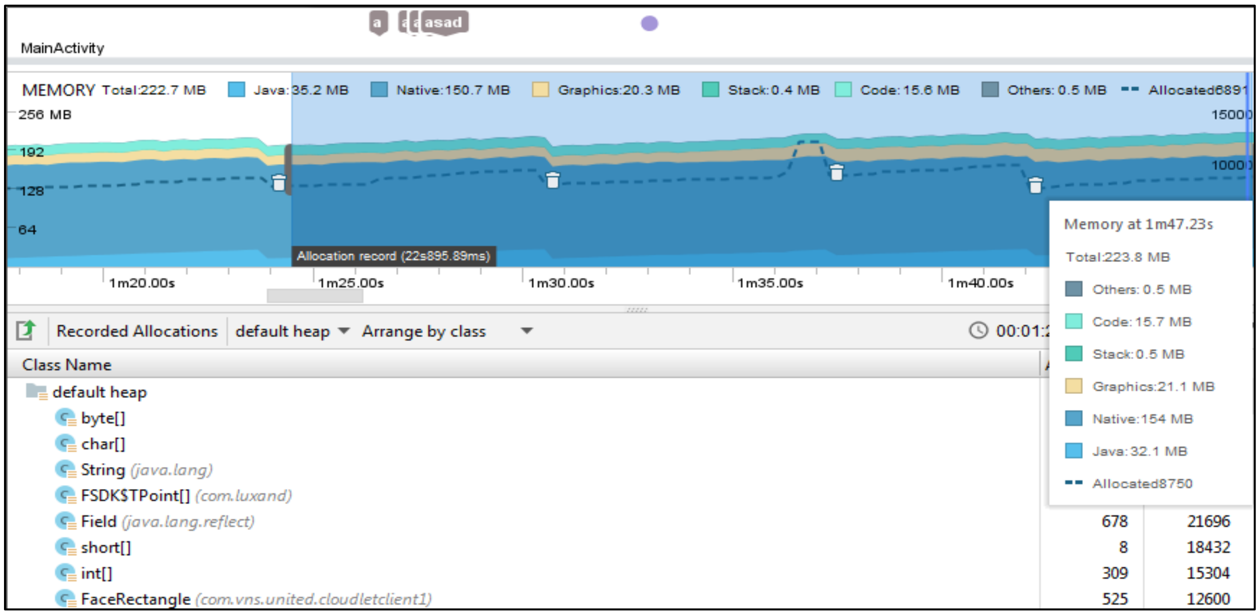


Figure 17 - CPU consumption on offloading task to Cloudlet

5.3.5 RAM Consumption

The consumption of RAM while executing the task on mobile is 223.8 MB as shown in Figure-18. It has been observed that RAM consumption is significantly high in this case.

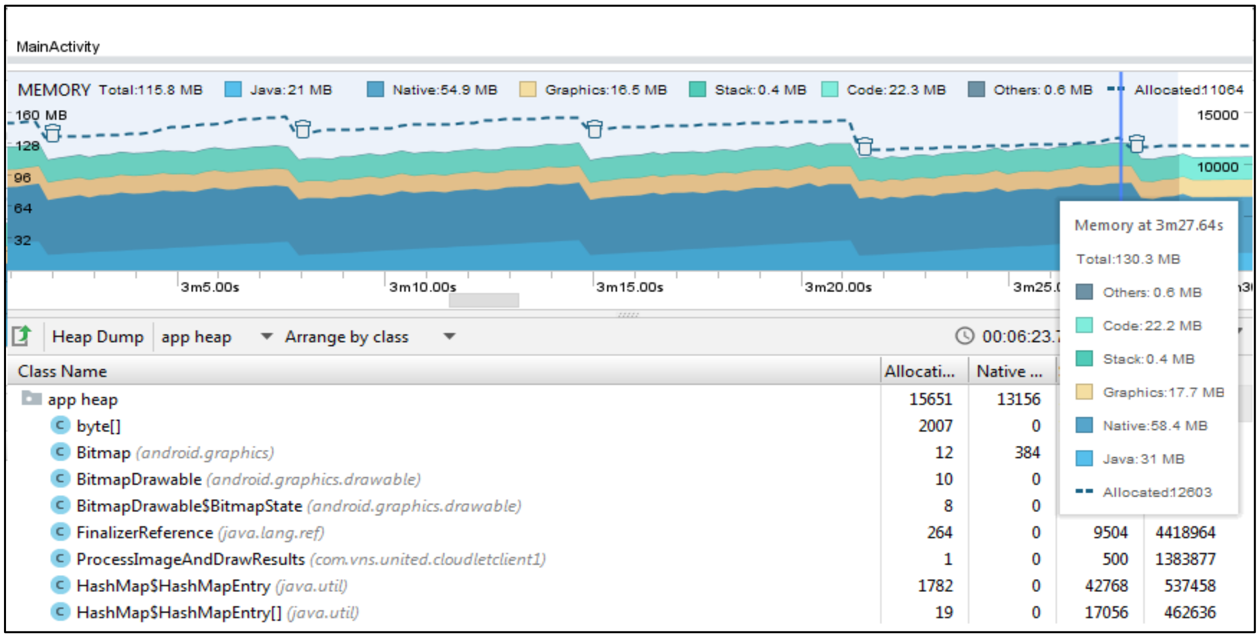


Memory at 1m47.23s
Total:223.8 MB
Others: 0.5 MB
Code: 15.7 MB
Stack:0.5 MB
Graphics:21.1 MB
Native:154 MB
Java:32.1 MB
Allocated8750

Class Name		
default heap		
byte[]		
char[]		
String (java.lang)		
FSDK$TPoint[] (com.luxand)		
Field (java.lang.reflect)	678	21696
short[]	8	18432
int[]	309	15304
FaceRectangle (com.vns.united.cloudletclient1)	525	12600

Figure 18 - RAM Consumption on Mobile

The RAM consumption while offloading the task to the cloudlet is shown in Figure-19. In this case the RAM consumption is only 130.3 MB which is about 94mb less then mobile case.



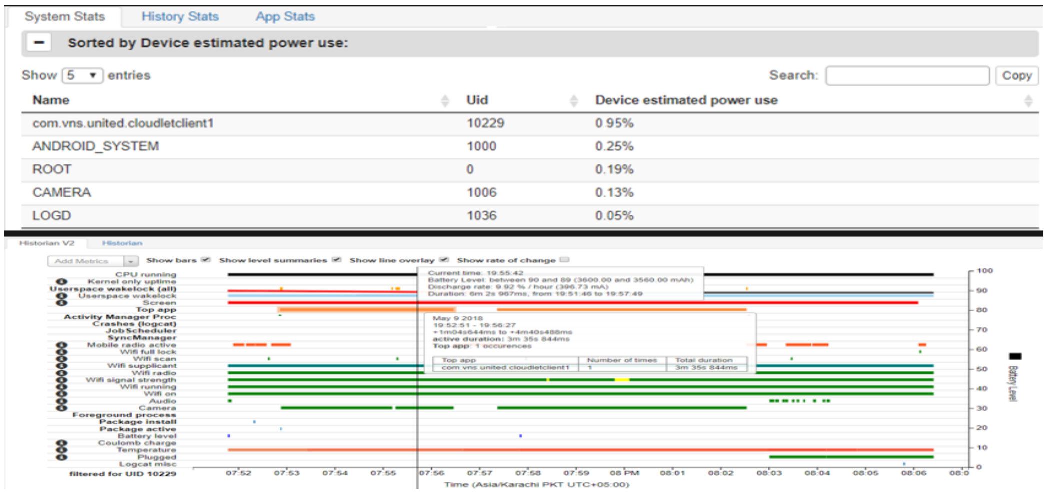
Memory at 3m27.64s
Total:130.3 MB
Others: 0.6 MB
Code: 22.2 MB
Stack:0.4 MB
Graphics:17.7 MB
Native:58.4 MB
Java: 31 MB
Allocated12603

Class Name	Allocati...	Native ...		
app heap	15651	13156		
byte[]	2007	0		
Bitmap (android.graphics)	12	384		
BitmapDrawable (android.graphics.drawable)	10	0		
BitmapDrawable$BitmapState (android.graphics.drawable)	8	0		
FinalizerReference (java.lang.ref)	264	0	9504	4418964
ProcessImageAndDrawResults (com.vns.united.cloudletclient1)	1	0	500	1383877
HashMap$HashMapEntry (java.util)	1782	0	42768	537458
HashMap$HashMapEntry[] (java.util)	19	0	17056	462636

Figure 19 - RAM Consumption on offloading task to Cloudlet

5.3.6 Battery Consumption

We analyzed the battery consumption of mobile application using Battery stats and Battery Historian. It is also an android tool that collects battery dump data using adb and then we can visualize the dump data using battery Historian. We performed 10 readings and in each reading we use 5 images to train and processing. It has been observed that battery consumption is 0.95% while executing the task on mobile as shown in Figure-20.



System Stats | History Stats | App Stats

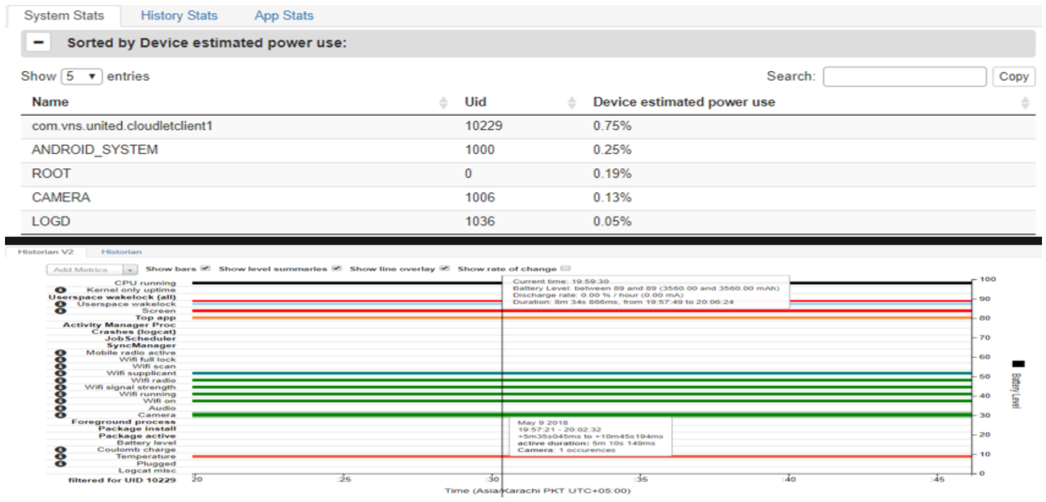
Sorted by Device estimated power use:

Show 5 entries   Search: Copy

Name	Uid	Device estimated power use
com.vns.united.cloudletclient1	10229	0.95%
ANDROID_SYSTEM	1000	0.25%
ROOT	0	0.19%
CAMERA	1006	0.13%
LOGD	1036	0.05%

Figure 20 - Battery Consumption on mobile

The battery comuption for task execution on cloudlet is shown in Figure-21. We observed that battery consumption was 0.75% .It is 20% less then mobile case. The results shows that mobile app uses a small amount of CPU, RAM and battery while offloading the task to the cloudlet. So we can save significant amount of mobile resources which can help to increase mobile life and aid resource poor mobile devices.



System Stats | History Stats | App Stats

Sorted by Device estimated power use:

Show 5 entries   Search: Copy

Name	Uid	Device estimated power use
com.vns.united.cloudletclient1	10229	0.75%
ANDROID_SYSTEM	1000	0.25%
ROOT	0	0.19%
CAMERA	1006	0.13%
LOGD	1036	0.05%

Figure 21 - Battery Consumption on offloading task to Cloudlet

CHAPTER 6: Conclusion and Future Work

6.1 Conclusions

Mobile technology is expanding rapidly, mobile industry offers high quality devices with greater processing power, good connectivity and fast memory etc. But still mobile devices lack in computational intensive tasks. Cloudlet computing is proposed as a solution to resource scarcity of mobile devices in which the computational intensive tasks are offloaded to the nearby cloudlet. These computational tasks get executed inside the cloudlet with the assistance of resource rich devices and the results are given back to the phone device.

In this thesis we discuss the two main concepts: Cloudlet computing and Mobile task offloading. We analyze different existing cloudlet frameworks and task offloading techniques and investigates different offloading issues that are still a challenge in Mobile Cloud computing. Study shows that there are two main approaches for task offloading: static offloading and dynamic offloading. There are variety of approaches but all of them aim to meet the same objective which is to improve the mobile phone capabilities by saving mobile resources, saving energy, to minimize execution cost and reducing response time.

We presented a mobile-Cloudlet architecture called "A dynamic framework to support offloading under resource constrained environment" as a platform to offload the target task "face recognition". This architecture is designed to automatically discover the nearby cloudlet server, get information of the server resources and dynamically decide between local execution and cloudlet offloading. We examine the energy, CPU and RAM consumption of the mobile device while performing the face recognition task on mobile and offloading to the cloudlet. Our results shows that we can save mobile resources by offloading the task to the nearby cloudlet.

In this thesis we analyze the performance of mobile device with respect to execution time. We found that task execution time depend upon different variables like available bandwidth between mobile and cloudlet, processing capacity of mobile and cloudlet, size of data to be offloaded to the cloudlet and communication latency etc. Results shows that we can decrease cloudlet execution time by increasing server resources and using high broadband bandwidth. We examine how execution time vary with processing capacity of mobile and server. In the second aspect we analyze the resource utilization of mobile device. We found that mobile device consumes more resources while performing the task on mobile. So it is beneficial to offload the task to the cloudlet. Mobile

Cloud computing can provide cost effective and efficient data management. We can decrease the execution cost and save energy. The architectures which can offload the intensive mobile tasks to the cloud will play an important role in future to enhance the lifetime and efficiency of mobile devices.

6.2 Future Work

Our proposed framework has feature enhancement capability for future. Offloading decision making parameters such as execution and response time, energy utilization, bandwidth are hard to measure. Therefore it will be further investigated that how these parameters can be estimate and measured. The proposed architecture evaluated only against the time based offloading decision, we can explore different offloading models including energy and CPU consumption of mobile devices.

Bibliography

[1] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon and W. Heinzelman, *Cloud-Vision: Real-time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture,* 2012.

[2] M. Satyanarayanant, Z. Chen, K. Hat, W. Hut, ,. W. Richtert and P. Pillai, *Cloudlets: at the Leading Edge of Mobile-Cloud Convergence,* 2014 6th International Conference on Mobile Computing, Applications and Services (MobiCASE), 2014.

[3] V. Praseetha, A. Bansal and S. Vadivel, *Mobile-Cloudlet Face Recognition: Two Different Approaches,* Journal of Computers, Volume 13, Number 1, January 2018.

[4] G. Yang and T. S. Huang, *Human face detection in a complex background*, 1994.

[5] H. Patel, *Face Recognition Using Mobile Cloud Computing,* 2015.

[6] F. Airton Silva, P. Maciel, E. Santana, R. Matos and J. Dantas, *Mobile cloud face recognition based on smart cloud ranking,* Springer, Computing Volume 99, March 2017.

[7] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, *The Case for VM-based Cloudlets in Mobile Computing,* Pervasive Computing , 8(4), 14-23, 2009.

[8] K. Ha, P. Pillai, W. Richter, Y. Abe and S. Mahadev, *Just-in-Time Provisioning for Cyber Foraging,* Proceeding of the 11th annual international conference on Mobile Systems, Applications, and Services-MobiSys '13, Taipei, Taiwan., 2013.

[9] T. Taleb and A. Ksentini, *Follow Me Cloud: Interworking Federated Clouds and Distributed Mobile Networks,* IEEE, 2013.

[10] S. Wang, K. Chan, R. Urgaonkar, T. He and K. K. Leung, *Emulation-Based Study of Dynamic Service Placement in Mobile Micro-Clouds,* IEEE, 2015.

[11] J. Y., Tawalbeh L., Ababneh F and Dosari F., "Resource Efficient Mobile Computing Using Cloudlet Infrastructure," in *IEEE Ninth International Conference on Mobile Ad-hoc and Sensor Networks*, 2013.

[12] Y. Li and Wenye Wang, "The unheralded power of cloudlet computing in the vicinity of mobile devices," in *IEEE Global Communications Conference*, 2013.

[13] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, Ranveer Chandra and Paramvir Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *Association for Computing Machinery*, 2010.

[14] S. I. P. M. M. N. a. A. P. B.-G. Chun, "CloneCloud: elastic execution between mobile device and cloud," in *sixth conference on Computer systems, EuroSys*, 2011.

[15] A. A. P. H. R. M. a. X. Z. S. Kosta, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *IEEE INFOCOM*, 2012.

[16] G. M. S., J. D. A. , M. S., M. Z. M. and C. X., "COMET: code offload by migrating execution transparently," in *10th USENIX conference on Operating Systems Design and Implementation*, 2012.

[17] P. Angin and B. Bhargava, "An Agent-based Optimization Framework for Mobile-Cloud Computing," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications,* 2013.

[18] R. K. Balan, M. Satyanarayanan, S. Y. Park and T. Okoshi, "Tactics-based remote execution for mobile computing," in *1st international conference on Mobile systems, applications and services*, 2003.

[19] D. Kovachev and R. Klamma, "Framework for Computation Offloading in Mobile Cloud Computing," in *International Journal of Artificial Intelligence and Interactive Multimedia*, 2012.

[20] R. Kemp, N. Palmer, T. Kielmann and H. Bal, "Cuckoo: a Computation Offloading Framework for Smartphones," Springer Berlin Heidelberg, 2012.

[21] Luxand, Luxand FaceSDK: Face Detection and Recognition Library Developer's Guide, 2018.

[22] F. Schroff, D. Kalenichenko and J. Philbin, "A Unified Embedding for Face Recognition and Clustering," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[23] "Openface," 2016. [Online]. Available: https://cmusatyalab.github.io/openface/.

[24] V. Praseetha and V. S., *Mobile Code Offloading: A Review*, IJETAE, 2016.

[25] T. Baltrusaitis, P. Robinson and L. Philippe Morency, "OpenFace: an open source facial behavior analysis toolkit," in *IEEE Winter Conference on Applications of Computer Vision*, 2016.

[26] "Opencv," [Online]. Available: https://docs.opencv.org/ref/master/d7/d9f/tutorial_linux_install.html. [Accessed 2018].