

Scalable and Dynamic Offloading of Resource Rich Mobile Application

MSc Research Project
Cloud Computing

Akash Srinivasan
Student ID: x19101279

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Akash Srinivasan
Student ID:	x19101279
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	17/08/2020
Project Title:	Scalable and Dynamic Offloading of Resource Rich Mobile Application
Word Count:	7420
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	16th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
------------------------	--

Signature:	
------------	--

Date:	
-------	--

Penalty Applied (if applicable):	
----------------------------------	--

Scalable and Dynamic Offloading of Resource Rich Mobile Application

Akash Srinivasan
x19101279

Abstract

There has been a recent growth in the usage of smartphones and complex applications. The hardware specifications of these smartphones vary but the computing complexity in real-time application such as image processing, face recognition, live stream gaming keeps increasing. This results in a bottleneck on the CPU utilization of the mobile device. To eradicate this, in this research proposal a framework has been proposed that uses offloading technology and combines the edge device and the cloud server for offloading. A novel algorithm known as resource provisioning considers the network status of the device and decides on the offloading location. An android application with Fibonacci series and image processing functionalities is developed to evaluate the experiment. Based on the execution time in the three environments as the parameter for evaluation, the results indicate that the execution time drastically reduces when performing offloading compared to the local execution.

1 Introduction

In recent years, there has a tremendous increase in the usage of smartphones, tablets, and so on. With the advancements in the hardware specifications of smartphones, high number of computing-intensive mobile applications capable of performing many complex tasks are being used. Real-time applications such as live gaming, video streaming, face recognition, and so on utilize the CPU, battery power, RAM of the mobile device at a much higher rate than the normal utilization rate. Such complex applications buffer in rendering the application in the user interface because of the hardware limitation of the mobile device as compared to the requirement of the mobile application.

Research has been done to eradicate this problem by offloading the computing-intensive part of the application to the cloud server or to the edge device and produce the output in the mobile device. Research works such as (Akherfi et al.; 2018) provides a survey on various offloading mechanisms and offloading techniques that can be used. There have been research works (Lin et al.; 2019; Wang et al.; 2019; Abbas et al.; 2017) focussing on partitioning the application, allocating the tasks, and offloading them to the edge devices. Research works such as (Flores et al.; 2018) have also focussed on performing application partitioning and offloading to the cloud server. Yet, very little attention has been given in integrating the cloud server and the edge device in a framework and performing offloading based on the device parameters.

Research Question: Computing-intensive part of the application can be migrated to a remote machine for processing but is there a way to integrate the edge device and the

cloud server in a framework for offloading based on the device parameters. The objective of this research work is to integrate both the edge device and the cloud server in a single framework and perform offloading based on the network parameter of the device. An algorithm known as resource provisioning is proposed which takes into consideration the network status of the device and based on which the offloading location is decided. A laptop within the same network of the mobile device is used as the edge device and an Amazon EC2 instance is used as the cloud server. Furthermore, to evaluate this research work, an android application is built with Fibonacci series and image processing functionalities, and the computing part of the application is executed on the local device, edge device, and the cloud server. The evaluation is based on the execution time parameter on all the three environments.

The rest of this paper is structured as follows. Section 2 details the related concepts and background works in this area. Sections 3 provides the methodology of this research. Section 4 describes the design specification used in this research. Section 5 details the implementation factors in this research. Section 6 entails the evaluation of this research work, and Section 7 provides the conclusion and the future work of this research.

2 Related Work

2.1 Cloud Server Based Offloading

In (Goudarzi et al.; 2017), they propose a fast hybrid multi-site computation offloading methodology, which aims to find a feasible offloading solution. The proposed methodology involves the usage of two algorithms which are the branch and bound algorithm and the PSO algorithm. The proposed architecture involves three modules namely monitor, modeler, and decision planner. The monitor collects real-time information about the mobile phone regarding battery status, CPU utilization, and the transmission rate. The individual units in the application are analyzed using a weighted relation graph and the result is given as an input to the decision-making module, which uses the two algorithms to find a feasible offloading solution.

The focus of the research is offloading to the cloud server and the algorithm helps in finding a feasible solution in terms of application partitioning and offloading to the respective cloud server. This offloading mechanism results in adding the transmitting and receiving time to the total execution time. Performing edge offloading might reduce this transeiving time and enhances security as well.

(Kwon et al.; 2016) research work focuses on performing offloading to cloud servers based on the prediction analysis using the proposed algorithm. The proposed mechanism predicts the energy consumption, memory state, and the CPU utilization of the individual units in the application. The individual units are offloaded to the cloud server and prediction analysis inputs are mapped to those units to predict the above-mentioned parameters and based on the output obtained, the decision-making module decides on the offloading of the application during the run-time. To experiment on the proposed mechanism, applications such as chess, face detection, and invaders are used, and promising results are obtained.

The prediction analysis mechanism provides more clarity on what to offload but this mechanism could involve offloading to edge devices as well to achieve even better results in the execution time compared to the transeiving time involved in offloading to the cloud server.

(Flores et al.; 2018) research work performs offloading on the cloud server based on the offloading framework EMCO constructed using the rich datasets regarding the energy consumption and CPU utilization. Based on the analysis performed by the analysis module for the method, the offloading is decided. The toolkit produces two output for local execution and cloud execution when compiling an application and the results from the cloud execution is cached and the results are instantly sent when the same application is offloaded. The AutoScaler module in the toolkit enables the cloud resources to be scaled dynamically based on the requirements of the offloading application.

The granularity of the offloading takes place at the application level and not the method or thread level. This incurs additional time for the application to be compiled in the cloud server. Since the research also focuses on the execution time, the edge server could also be used with the cloud server for smaller transceiving time and better security.

2.2 Edge Devices Based Offloading

In (Saha et al.; 2016), they propose a method to offload the resource-intensive mobile application to the devices called donors connected in the cloudlet using an algorithm that takes into consideration the available battery, the availability of the donor device in the network. The algorithm considers a donor device based on the condition that the donor device must have more than 40 battery and signal strength of -80db. Also, the modules in the resource-intensive application are broken down into smaller modules based and structures into a directed graph based on the dependency level of the modules. These modules are offloaded to the various donors available and the results are fetched back.

This research works well for big integer and prime number calculation application, but real-time applications such as augmented reality and video games will lag because of the dependency in executing the parent module before executing the child module in the donor devices. Since the donor devices are also mobile devices, a more feasible solution would be to offload to the cloud servers, rather than performing the offloading methods in the donor mobile device due to the resource constrain.

(Elgendy et al.; 2019) research work focuses on performing offloading from mobile units to mobile edge computing servers with additional security measures while performing offloading to the edge servers over a wireless medium. An algorithm is developed which obtains the real-time statistics of the mobile unit and performs the offloading. The statistics regarding the mobile device are obtained in a tuple containing information regarding the battery, the number of CPU cycles required data size, and the completion deadline. Apart from this, the AES cryptographic technique is used on the data that is transmitted wirelessly. To test this proposed mechanism, a face detection application is developed and tested and the experimented shows a promising result in terms of offloading and security.

For larger applications such as live streaming and video games, it is difficult to employ this security mechanism. Also, the offloading should extend to cloud servers and the same security mechanism could be deployed in cloud server offloading.

(Wu et al.; 2018) paper performs the mobile code offloading in edge servers by modifying the unikernal and using a new run-time in the edge servers and measures the parameters such as execution time, performance, and the energy consumption of the application and compares them with the traditional virtual machine and container-based code offloading. Unlike traditional unikernal which is specialized for one application at a

time, the proposed rich unikernal will be able to accommodate multiple applications during the run time. Traditional android libraries are integrated within the OS used. This benefits in reduced overhead boot times and compilation time for multiple applications.

Using the ThinkAir framework used for offloading, applications such as Chess game, face detection, VirusScan, and Linpack benchmarking are executed using offloading in traditional VM and using rich unikernal and the parameters such as execution time, performance, and the energy consumption are compared and promising results are obtained. This could be further enhanced if cloud server offloading could be integrated.

(Alelaiwi; 2019) research work focuses on performing offloading based on the response time obtained from the deep learning model to the fog nodes containing edge devices. The presented architecture includes fog access points and fog access point controller. Inter fog communication is achieved by the usage of the evolved packet data gateway which further connects with the AAA module (Authentication, Authorization, and Accounting server) to maintain security integrity. Using the prediction time obtained from the deep learning model, the node with the shortest response time is selected for offloading. The greedy layer-wise training is provided to the model to predict the future response time which is based on the input such as capability, memory, and bandwidth of the requests provided to the model.

Simulation-based results are obtained using MATLAB with the datasets obtained using the queueing theory. The proposed model is not illustrated with a real-time application and the research does not explain the location of the implementation of this architecture.

(Tao et al.; 2017) research focuses on performing offloading to the mobile edge computing server based on the energy requirements and the CPU utilization of the individual mobile devices. When offloading for an application is to be performed, the energy requirements for executing the method or the application in the mobile device and the CPU utilization on the mobile device are known. Based on the status of the mobile device regarding the battery consumption and the CPU cycles required, the offloading is decided, and the required capacity is allotted in the edge computing server. Determining the requirements in the mobile device of the application is a crucial factor in this research for delegating and executing the task. The experiment in this research is performed by determining the capacity of the mobile device, the operation is executed, and the capacity of the edge computing server to allocate resources for multiple mobile devices.

This research work does not provide an algorithm for problem formulation and computing the offloading. No real-time applications were used for this experiment. For a highly computing-intensive application such as augmented reality and face detection, determining the resources required might not be a feasible solution as the requirements might change dynamically. Also offloading to cloud might be a more feasible solution to highly perform resource-intensive methods or applications.

2.3 Cloud Servers and Edge Devices Based Offloading

(Zhao et al.; 2019), they perform offloading to Small Base Stations (SBS) directly and to Mobile Edge Computing servers (MEC) through wi-fi access points based on the traffic predictions. A multi-Long Short Term Memory Model (mLSTM) traffic predictor was constructed and it is used to predict the traffic of the data flow in SBS. The data that is used for offloading is cached so that when the same offloading is performed second time, the cached data can be instantly retrieved. The traffic predictor is used to predict the

traffic flow in the SBS and based on the data cached, offloading priority can be predicted. Offloading of data with low priority takes place in the MEC through wi-fi access point and high priority offloading methods takes place in the SBS.

To evaluate the proposed methodology, dataset of usage detail records of a cellular company from china is used. The traffic prediction and the task delegation for offloading is tested using this dataset. Real-time applications have not been tested using this methodology. Involving cloud offloading might be a feasible solution to offload highly resource intensive applications.

(Jungum et al.; 2020) paper focuses on performing application partition for code offloading using Analytic Hierarchy Process (AHP) and TOPSIS technique. For a highly resource-intensive application, the partitioning of the application and the task delegation is based on the weightage obtained for the following parameters such as bandwidth, performance, availability, and security. AHP technique is used to predict the weightage of the following parameters and the TOPSIS technique is used to rank the parameters based on the weightage. Based on the following output, partitioning and task delegation takes place.

This research paper does not provide an algorithm for the proposed method and does not perform cloud or edge offloading with real-time applications. Moreover, the predictions should also consider additional parameters such as the feasible network conditions for cloud and edge offloading and provide a more feasible partitioning strategy.

(Zhou et al.; 2017) paper performs code offloading on the cloud server, edge devices, mobile ad-hoc devices using various mediums such as wi-fi, wi-fi direct, Bluetooth, and mobile data based on the status of the mobile device and the availability of the resources. The context monitor module profiles the program for methods to be offloaded, the network profiler scans the network and checks for the resources available, and the device profiler checks the statistics of the device in terms of the CPU utilization and the energy consumption. Based on the output of the context monitor, the algorithm in the decision engine decides on the offloading location. This research work is compared with the existing offloading technique ThinkAir based on the execution time, CPU utilization, and energy consumption.

This research performs offloading based on the annotations for the methods denoted by the developers. This leaves the overhead on the developers to predict the methods to be offloaded while building the application.

2.4 NFC and Bluetooth Offloading

(Chatzopoulos et al.; 2020) research work focuses on performing offloading of methods based on NFC communication protocol. To eradicate the unidirectional communication in NFC based transmission, Host-Based card emulation is used for bidirectional communication. With this usage, no tapping is required to perform offloading with multiple data transfer. The method to be offloaded is sent to the offloadee device and the result is sent back using this mechanism. The proposed framework is tested with applications such as NQueens, RSA, face detection, and data transfer. This provides security for the methods being offloaded and faster transmission of data but leaves the overhead for the devices to be very near compared to Bluetooth and wi-fi. Also, the offloadee device is required to be equipped with Host-Based card emulation for bidirectional communication.

(Lin et al.; 2020) research work focuses on performing offloading to other mobile devices based on the proposed framework known as Circa. The proposed framework

uses the iBeacon technology in the iOS platform that leverages Bluetooth technology with added features such as long-distance transmission and faster transmission of data. The mobile device that offloads the method, detects the neighboring mobile device with iBeacon framework which falls under the vicinity of the offloading mobile device. To ensure the detected mobile device is a reliable connector, the history of the duration of the mobile device being in the vicinity of the offloading device is considered with the battery status of the device. The proposed algorithm is framed in such a way where if the device is trusted to be a reliable connector, the method is offloaded only to that device and if not, the same method is offloaded to other mobile devices as well which are detected in the same vicinity.

The proposed framework only works on the iOS platform. Despite offloading, the processing device is also a mobile device with limited resources so the benefits of offloading would be extremely low as we also consider the transmitting and the receiving time. Furthermore, it would not be a feasible solution to offload highly complex mobile applications.

2.5 Survey Papers

(Flores et al.; 2014) research performs a survey on the existing offloading techniques with the cloud, compares them with task delegation. The research experiments by creating five applications such as Fibonacci series, NQueens, matrix multiplication, quicksort, and bubble sort and performs offloading across different amazon EC2 instances to check the execution time. The offloading in those applications is based on annotations which are dynamically called during the run time. Apart from this, experiment on stenography is also performed with task delegation and push notifications and it is compared with annotation-based cloud offloading. Various research works based on cloud offloading and techniques used for offloading using the API provided by the cloud vendors and protocols that could be used for offloading are analyzed.

The research work performs a survey and conducts an experiment based on the cloud offloading focusing on the execution time, but the research did not explore edge offloading and its potential benefits compared to the cloud offloading.

(Nguyen and Dressler; 2020) research work provides a detailed survey on mobile cloud computing regarding the application partitioning, profiling the device dynamically, the existing offloading algorithms, remote procedure calls(RPC) used for offloading to cloud servers, usage of various types of resources for offloading and the challenges about mobile cloud computing. The existing works in the device profiling domain such as obtaining the battery and the CPU status of the mobile device, available network, and bandwidth conditions, analyzing the application for static and dynamic offloading, providing these inputs to the decision engine for offloading decision is explained. Furthermore, the existing algorithms used in the decision engine such as fuzzy control, Bayesian networks, learning agents, game theory, deep learning, and other such algorithms are explained. From the research point of view, this research survey will greatly benefit researchers researching mobile cloud computing.

(Gu et al.; 2018) paper provides a survey on the existing offloading mechanisms and explains the research works in these domains. Various offloading techniques and offloading infrastructure such as cloud servers, edge devices, mobile ad-hoc networks, fog computing are explained with its respective research works. The application profiling such as collecting the statistics of the running application dynamically and the level of

granularity in which the application can be offloaded are explained with its respective research works. Furthermore, the traditional frameworks that are used in offloading such as MAUI, CloneCloud, ThinkAir, are explained.

Apart from this, real-time applications that have benefited in the past research works using offloading such as face recognition, chess game, video encoding, speech recognition, and the challenges about mobile cloud computing are explained.

From the conducted literature review, performing offloading in the previous research works have been classified into various categories. There have been surveyed papers that elaborate on the various techniques that can be adopted for offloading. Research works conducted perform offloading either to the edge devices or to the cloud server. Offloading has been performed using NFC and Bluetooth to the neighboring devices in the network. If the offloading is performed using the edge devices and the cloud servers, annotation-based programming is required to identify the potential candidate for offloading.

The proposed framework uses the state-of-the-art technique in offloading by taking advantage of both the Edge devices and the Cloud servers and using network profilers to check for the offloading feasibility and decides on the location of the execution.

3 Methodology

The various research methods that have been adopted in this research work are presented here. Offloading performed in this research work involves offloading to the edge device and offloading to the cloud server. The procedure for performing the offloading in both locations is explained. The evaluation methodology used to evaluate this research work is briefed.

3.1 Application used for the research procedure

To validate the research proposal Android mobile application is developed for generating the Fibonacci series and performing image processing. Application is provisioned with the resource provisioning algorithm. Based on the network connectivity of the mobile device, offloading decisions are made as prescribed in the algorithm. If the decision is made for performing local execution, the Fibonacci series is generated locally for the input number provided.

For the image processing part of the application, 13 images of 5 classifications which are obtained from kaggle¹ are stored locally. The trained model that is used to classify the images is obtained from github². This trained model uses the convolution neural network algorithm. One among the thirteen is chosen for processing and when the classify image button is clicked, the onClick function checks for the network status of the device and the data is sent to the edge device or to the cloud server for offloading. 13 images that are used for image processing are stored in the edge device and the cloud server and the processed data is sent back to the mobile device in the JSON format and the results are displayed. Volley API is used for handling the network requests.

¹<https://www.kaggle.com/hsp6498/waste-dataset>

²<https://github.com/vasantvohra/TrashNet>

3.2 IP addresses of the machines

In figure 1, the red box denotes the URL of the localhost connection for edge devices. The condition checks for the availability of the wifi network and if the condition is true, offloading to edge device takes place. The yellow box specifies the IP address of the cloud server. In the IF condition, if the edge device is not available, then offloading to the cloud server takes place through wifi.



```
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195

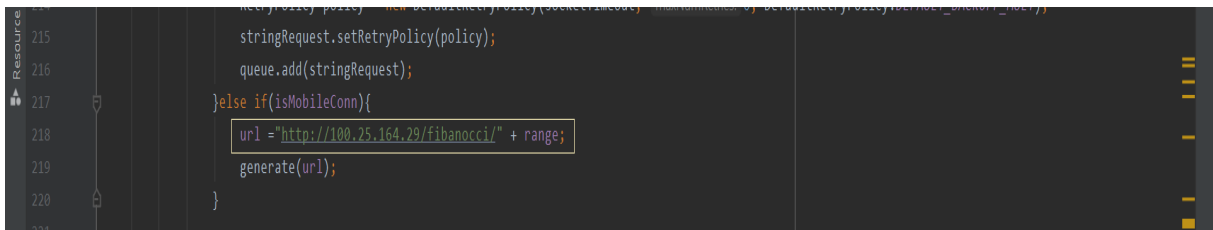
if(isWifiConn){
    url = "http://192.168.0.102:5000/";
    //url = "http://192.168.43.43:5000/" + range;

    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        (response) -> {
            // Display the first 500 characters of the response string.

            edge_status = true;
            Log.d( tag: "Edge", msg: "Edge status:" + edge_status);
            url = "http://192.168.0.102:5000/fibonacci/" + range;
            generate(url);
        }, new Response.ErrorListener() {
            @Override
            public void onResponse(VolleyError error) {
                error.printStackTrace();
                edge_status = false;
                Log.d( tag: "Edge", msg: "Edge status:" + edge_status);
                if(isMobileConn) {
                    url = "http://192.168.43.43:5000/fibonacci/" + range;
                    generate(url);
                }else{
```

Figure 1: Specifying the IP address for the edge device using wifi connection

In figure 2, the yellow box specifies the IP address of the cloud server. The else part of the condition checks for the availability of the mobile network. If the condition is true, then offloading to the cloud server will take place. Upon both the conditions being false, local execution of the tasks will take place.



```
215
216
217
218
219
220
221

stringRequest.setRetryPolicy(policy);
queue.add(stringRequest);
}else if(isMobileConn){
    url = "http://192.168.43.43:5000/fibonacci/" + range;
    generate(url);
}
```

Figure 2: Specifying the IP address of the cloud server for mobile network

3.3 Offloading to the Edge Device in the network

To perform offloading to the edge device, a jupyter notebook in anaconda navigator is used. The tool is installed on a laptop that is connected to the same wifi network as the mobile phone. Flask application is built in the jupyter notebook using python for compiling the Fibonacci series and image processing. When the scripts are compiled, a localhost connection is created and waits for the application installed in the mobile device to send an input to the python scripts using the localhost connection. The input entered in the mobile application is sent to the edge device in a JSON format.

Once the input reaches the edge device, the python script compiles the code for the Fibonacci series or image processing and sends the string output to the client device connected using the localhost. To manage the network requests, volley API is used

for sending and receiving of the data. By offloading the input to the edge device, the computation performed on the edge device with the computational capacity of the edge device, and the results are sent to the mobile device. This procedure provides faster processing of data, the transceiving time of the data is also smaller due to the reduced distance compared to cloud offloading. Security of the data transmission is also enhanced as the data is offloaded to the nearby edge device using the localhost.

```

* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off

* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

```

In []:

In []:

Figure 3: Establishment of a localhost connection

The overall execution time of this model is sending the input to the edge device + executing the program in the edge device + sending the results back to the mobile device.

3.4 Offloading to the Cloud Server

Offloading to the cloud server takes place when the mobile contains the mobile data network. Python Machine Learning model is exposed as Rest Service using Flask. Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. We deploy our model in EC2 instance as Flask Application. Since Flask framework is not production ready, we use Gunicorn. The Gunicorn "Green Unicorn" is a Python Web Server Gateway Interface HTTP server. It is a pre-fork worker model, ported from Ruby's Unicorn project. The Gunicorn server is broadly compatible with a number of web frameworks, simply implemented, light on server resources and fairly fast. This makes the application more stable.

Applications hosted in the public domain should be more secure and stable to avoid unnecessary DDOS attacks. NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. NGINX is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption. We use NGINX HTTP server in front of Gunicorn to make our cloud device more secure and stable. Using the same technique in edge offloading, volley API is initially used to establish the connection and the data to be offloaded is sent in the JSON format. In the same way, the results are fetched back. When cloud offloading takes place, the total execution time is the time taken for the data to be transferred + time taken for the data to be executed + time taken for the data to be retrieved.

Mobile Device — *NGINX* — — — *Gunicorn* — — — *Flask*

Execution of Gunicorn is created as a service for smooth deployment of Flask application in cloud.

3.5 Evaluation methodology

The application is deployed on the mobile device. To evaluate the application, execution time is considered as the parameter. Since the resource provisioning algorithm decides on the offloading location based on the network status of the device, the application is tested in different network status on the mobile device. Three different workloads on the Fibonacci series and three different images on the image processing part of the application are used for testing. These workloads are deployed on the edge device and the cloud server and execution time of these workloads are considered. Apart from this, the local execution time of these workloads on the mobile device is also considered for evaluation, and based upon the results obtained the conclusion is drawn.

4 Design Specification

4.1 Resource Provisioning Algorithm

Various profilers are used in the device for achieving an optimal offloading. In this research, the program profiler and network profiler are used to obtain the context for offloading. The program profiler identifies the potential candidate to be offloaded and the network profiler checks the status of the network connectivity. Based on the current network status of the device, the location for offloading is chosen. IF loops and nested IF loops are used in the conditions of the algorithm. If the device contains a mobile data network, then cloud availability is checked and automatically `cloud_execution` takes place. If mobile data or cloud server is not available, then `local_execution` takes place. If the device is connected to the wifi network and a potential edge device is available, automatically `edge_offloading` takes place. If the edge device is not available, `cloud_execution` will be chosen and if both are not available, `local_execution` takes place.

4.2 Workflow of the algorithm

4.3 Architecture diagram of the framework

4.4 Context Monitor

Context monitor contains the program profiler and the network profiler. The program profiler identifies the candidate that needs to be offloaded. In the Fibonacci series part of the application, the input given to generate the Fibonacci series is the candidate to be offloaded. In the image processing part of the application, the image selected is the candidate to be offloaded. These candidates are identified by the program profiler. The network profiler checks the network status of the device. The output from the context monitor is provided as the input to the decision engine.

4.5 Decision Engine

The decision engine contains the resource provisioning algorithm. Based on the output provided from the context monitor, the decision engine performs the algorithm. Resource provisioning algorithm is framed in such a way where offloading takes place in cloud servers when only the mobile data network is available, and offloading takes place in edge device if the mobile device is connected only to the wifi network. If the edge device is

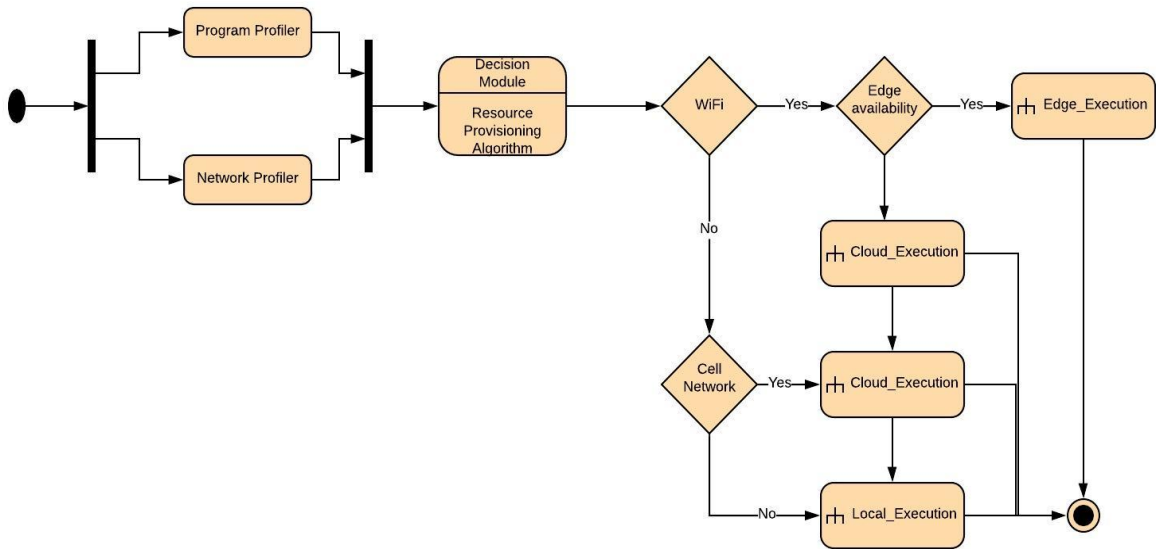


Figure 4: Workflow of the algorithm - Activity Diagram

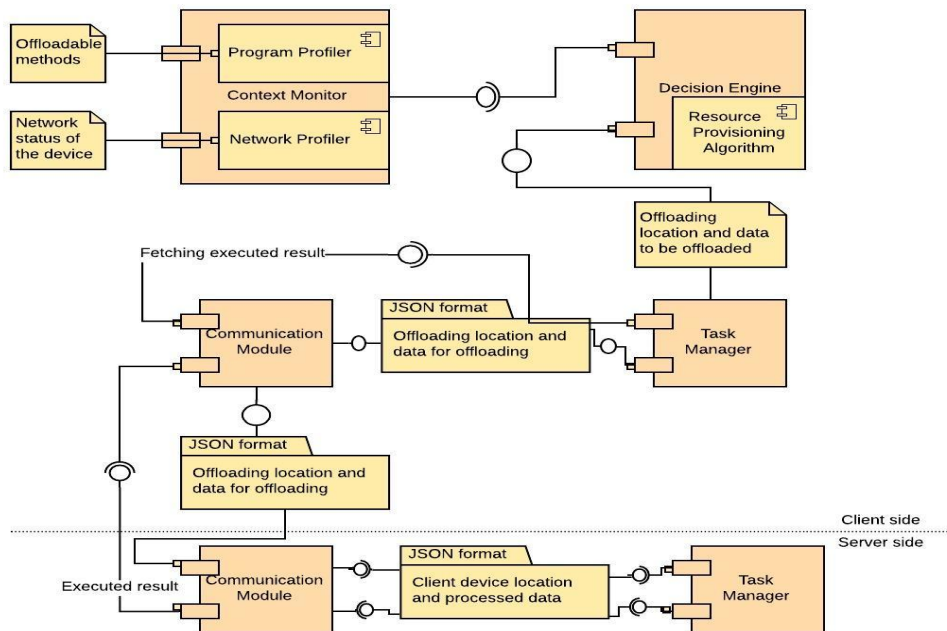


Figure 5: Architecture diagram of the framework - Component Diagram

not available or busy, then offloading takes place in the cloud server through the wifi connection. If the mobile is not connected to both the networks, then local execution takes place. The offloading location and the candidate for offloading are chosen by the decision engine. The details are sent to the task manager for offloading.

4.6 Task Manager

From the output generated from the decision engine, the tasks manager wraps the Fibonacci input number or the image to be classified in a JSON format. On the server-side, to compile the Fibonacci series or the image processing, python scripts are programmed on the jupyter notebook in anaconda navigator tool. Once the input is received on the server-side, the task manager unwraps them and provides them as the input to the python scripts. It then performs the execution and provides the result. These results are sent in the JSON format to the mobile device.

4.7 Communication Module

The communication module is responsible for establishing the connection between the client-side and the server-side. For edge offloading, the mobile device and the edge device should be on the same network. Using localhost, the connection is established. For both edge offloading and the cloud server offloading, volley API is used that sends and retrieves the data in JSON format.

5 Implementation

5.1 Algorithm

Algorithm 1: Resource Provisioning Algorithm

```

Result: Getdecision(context)
1 programprofiler ← methods ;
2 network profiler ← resource profiles
3 Inspect network condition
4 if only cell network = available then
5 |   check cloud availability;
6 if cloud = available then
7 |   return decision(cloud_execution);
8 else
9 |   return decision(local_execution);
10 end
11 else if only WIFI = available then
12 |   check cloud, edge availability;
13 |   return decision(edgedevice_execution);
14 else if edgedevice = busy then
15 |   return decision(cloud_execution);
16 else
17 |   return decision(local_execution);
18 end

```

The resource provisioning algorithm is embedded inside the application. The application considers the current network status of the device and arrives at an offloading decision. Offloading locations such as edge devices and cloud servers are considered when deciding for offloading. The algorithm is activated once the button for generating the Fibonacci series or the button for performing the image processing is clicked. Multiple IF loops are checked and if offloading cannot take place based on the network status of the device, local execution of the task takes place. This local execution cannot take place in the image processing section of the application due to the complexity in computation.

5.2 Deployment diagram of the framework

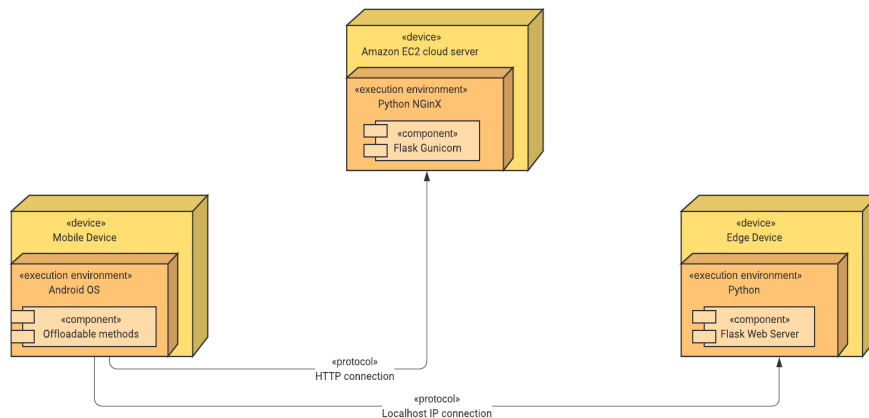


Figure 6: Deployment Diagram

Image Generated From: <https://www.lucidchart.com/>

The deployment diagram depicts the execution environment of the research framework including the software and the tools used for executing the application. The mobile device is connected to the cloud server using the HTTP connection protocol and it is connected to the edge device using the localhost connection. On the cloud side, the Nginx web server is used as an execution environment and flask application built using python is used to process the task. Gunicorn acts as the WSGI (Web Server Gateway Interface) for forwarding the requests to the application. On the edge device side, the flask application with Nginx web server is used as the component, and python scripts are used to process the task forwarded to the edge device and produce the result in the string format. Localhost connection using the IP address is used to connect to the edge device if the mobile device and the edge device is connected in the same network. The IP address of the edge device and the cloud server are posted as an URL in the algorithm inside the application in android studio for the requests to be forwarded.

5.3 Specification of the experiment environment

The table specifies the specifications of the research environment. To test the proposed research work, devices in three different environments i.e., the local device, the edge device, and the cloud server are used with various workloads on both parts of the application i.e., the Fibonacci series and the image processing part of the android application. The local device used is a Samsung Galaxy S7 Edge mobile device. The edge device used is a

Specifications of the experiment environment			
Specifications	Samsung Galaxy S7 Edge (Mobile Device)	HP Laptop (Edge Device)	Amazon EC2 Instance (Cloud Server)
CPU	Octa-core (4*2.3 GHz)	Intel i7 8th Gen	1 vCPUs
RAM	4GB	8GB	1GB
Chipset	Exynos 8890	-	-
OS	8.0.0	Windows 10	Linux
Tool	-	Anaconda	Nginx, Gunicorn, Flask
Language	-	Python	Python

HP laptop. Amazon EC2 instance is used as the cloud server. The specifications of the environments are provided in the above table.

6 Evaluation

To evaluate this experiment, the application is tested with three different workloads for the Fibonacci series and image processing. The execution takes place on the local device, edge device, and the cloud server. Based on the execution time taken on the three environments with three different workloads, the conclusion is drawn. Since the time taken to execute each workload fluctuates constantly, three readings are taken and the average of the three readings is considered as the final time consumption to execute the workloads.

Workloads: The workload for the Fibonacci series is provided in an incremental model i.e., for each experiment, the input provided to the Fibonacci series increases. For image processing, from the three categories of images used in this experiment, one image from each category is used in each experiment. Due to the computational complexity involved in processing images, it cannot be performed locally. It requires processing either by offloading to the cloud server or the edge device.

6.1 Experiment 1 / Fibonacci series workload as 3000 and glass image workload for image processing

The results denote that compared to local execution, offloading to edge devices or the cloud server drastically reduces the execution time.

Glass image is used for image processing in this experiment. There is not much of a difference in the time taken between cloud execution and the edge execution. Compared to edge execution, cloud execution takes lesser time for execution.

6.2 Experiment 2 / Fibonacci series workload as 4000 and paper image workload for image processing

The obtained results indicate that the execution time when offloading is hugely reduced when compared to local execution.

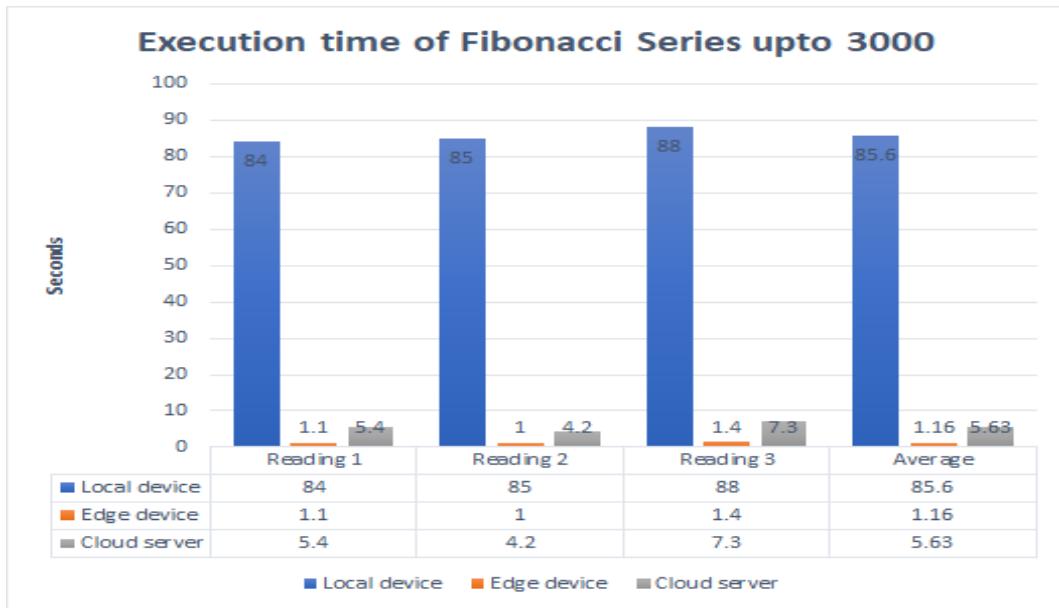


Figure 7: Time taken for 3000 Fibonacci series iterations in three environments

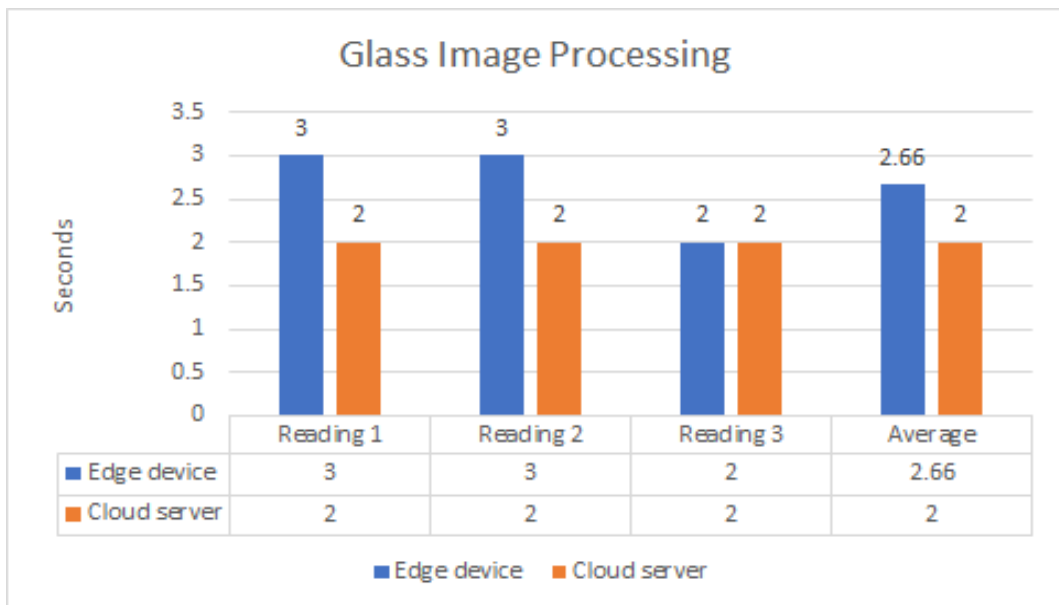


Figure 8: Time taken for Glass Image processing in the three environments

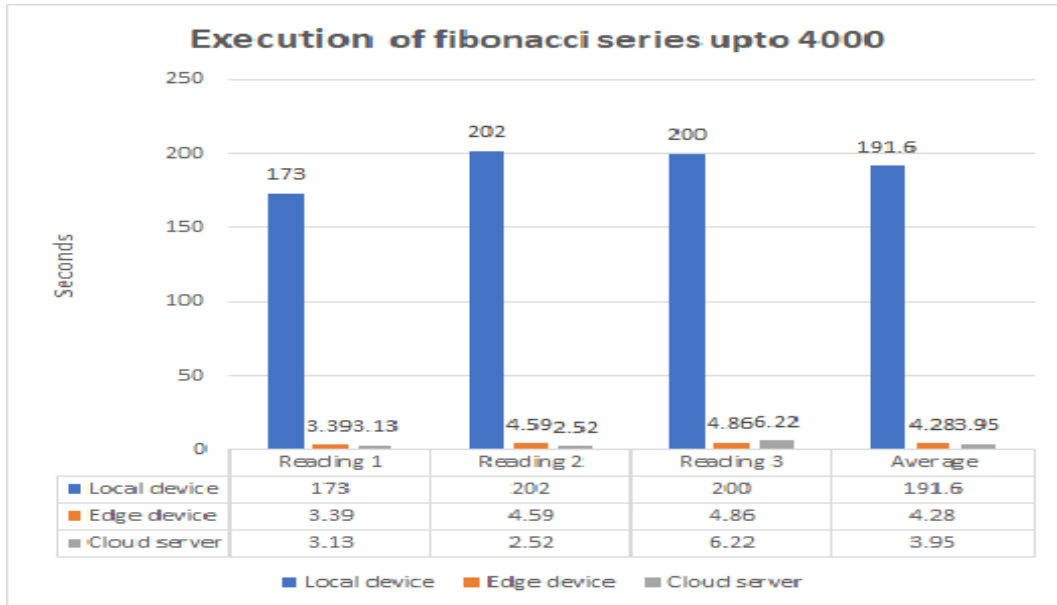


Figure 9: Time taken for 4000 Fibonacci series iterations in three environments

Paper image is used for image processing in this experiment. There is not much of a difference in the time taken between cloud execution and the edge execution. Compared to edge execution, cloud execution takes lesser time for execution.

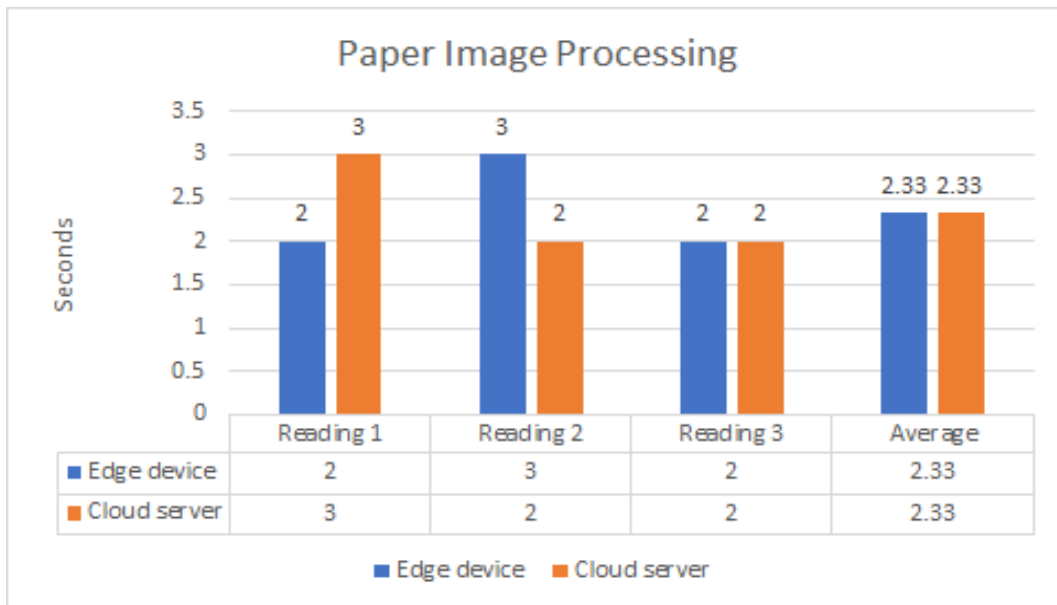


Figure 10: Time taken for paper Image processing in the three environments

6.3 Experiment 3 / Fibonacci series workload as 5000 and trash image workload for image processing

The time taken to execute this workload can be calculated in minutes when performing local execution, but when offloaded, the execution takes place in less than ten seconds.

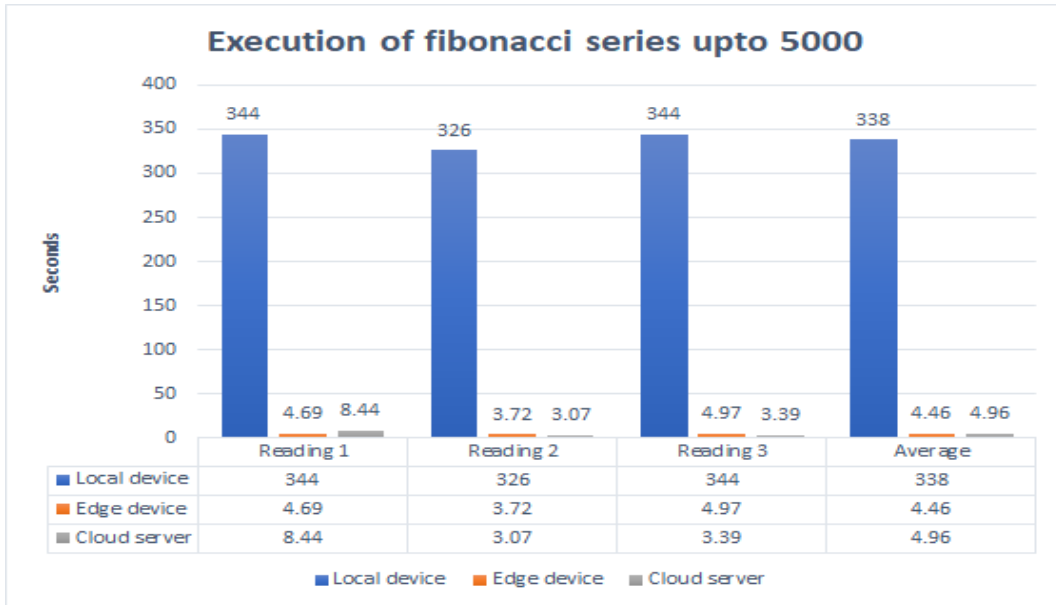


Figure 11: Time taken for 5000 Fibonacci series iterations in three environments

Here, the Trash image is used for image processing. The results depict that the time taken to execute image processing in the edge device and the cloud server is approximately the same.

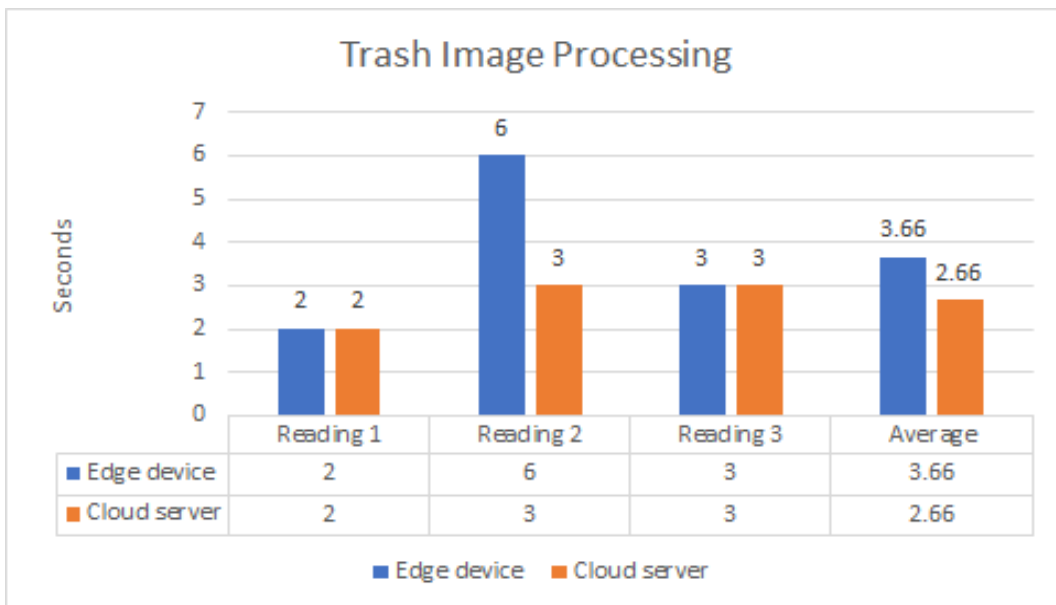


Figure 12: Time taken for Trash Image processing in the three environments

6.4 Discussion

This experiment is performed with three different workloads in the three experiments on both parts of the application. Based on the results obtained, it can be seen that the time taken to execute these workloads drastically reduces when offloaded either to the edge device or to the cloud server. Moreover, offloading to the edge device provides more

security as data is transceived within the network. The execution time when offloaded to the edge device is also lower compared to the cloud server due to the vicinity of the edge device from the proximity of the mobile device. Furthermore, the time taken to execute the workload in the cloud server can be even more reduced if the EC2 instance of higher hardware specification is used. Since free tier EC2 instance is used in this research, it takes more time to execute than advanced EC2 instance.

The execution time is calculated based on the computation time taken for the workload to be executed in both the edge device and the cloud server. But the results take more time to be displayed in the user interface of the mobile device than the computation time as the user interface needs to render a huge amount of result in the application mobile screen. This rendering time taken for the user interface to display the huge results could have been calculated, and also the CPU utilization and the battery consumption on three different environments could have been considered in this research.

7 Conclusion

In this research work, the framework for offloading the computing-intensive part of the android application is implemented. In this framework, the offloading location is decided based on the current network status of the device. A laptop in the network is considered as the edge device and Amazon EC2 instance is used as the cloud server. To evaluate this experiment, an android application is built with the Fibonacci series and image processing functionalities. The complex part of the application is offloaded to the edge device and the cloud server based on the network status of the device. Apart from this, local device execution of the Fibonacci series and image processing is also performed. The execution time is considered as a parameter to evaluate.

The obtained results indicate that performing offloading significantly reduces the execution time of the complex task as compared to the local execution. This also helps in reduced CPU consumption and prolonged battery life in the mobile device. This research framework is a comprehensive improvement in mobile task offloading considering the usage of the edge device, cloud server, and local execution based on the network status of the device.

7.1 Future work

This research work will be further extended to the optimization of the results obtained from the edge device and the cloud server. Apart from considering execution time as the parameter, the CPU utilization and the battery consumption is also to be considered as a parameter for evaluation in validating the result. Furthermore, dividing the task and concurrently executing the tasks on multiple edge devices for faster processing while offloading is also planned.

References

- Abbas, N., Zhang, Y., Taherkordi, A. and Skeie, T. (2017). Mobile edge computing: A survey, *IEEE Internet of Things Journal* **5**(1): 450–465. JCR Impact Factor : 5.863 (2018).

- Akherfi, K., Gerndt, M. and Harroud, H. (2018). Mobile cloud computing for computation offloading: Issues and challenges, *Applied Computing and Informatics* **14**(1): 1 – 16.
URL: <http://www.sciencedirect.com/science/article/pii/S2210832716300400>
- Alelaiwi, A. (2019). An efficient method of computation offloading in an edge cloud platform, *Journal of Parallel and Distributed Computing* **127**: 58 – 64. JCR Impact Factor : 1.815 (2018).
URL: <http://www.sciencedirect.com/science/article/pii/S0743731519300140>
- Chatzopoulos, D., Bermejo, C., Kosta, S. and Hui, P. (2020). Offloading computations to mobile devices and cloudlets via an upgraded nfc communication protocol, *IEEE Transactions on Mobile Computing* **19**(3): 640–653. JCR Impact Factor : 4.098 (2018).
- Elgendy, I. A., Zhang, W., Tian, Y.-C. and Li, K. (2019). Resource allocation and computation offloading with data security for mobile edge computing, *Future Generation Computer Systems* **100**: 531 – 541. JCR Impact Factor : 4.639 (2018).
URL: <http://www.sciencedirect.com/science/article/pii/S0167739X18328346>
- Flores, H., Hui, P., Nurmi, P., Lagerspetz, E., Tarkoma, S., Manner, J., Kostakos, V., Li, Y. and Su, X. (2018). Evidence-aware mobile computational offloading, *IEEE Transactions on Mobile Computing* **17**(8): 1834–1850. JCR Impact Factor : 4.098 (2018).
- Flores, H., Srirama, S. N. and Buyya, R. (2014). Computational offloading or data binding? bridging the cloud infrastructure to the proximity of the mobile user, *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, Oxford, UK, pp. 10–18.
- Goudarzi, M., Zamani, M. and Haghghat, A. T. (2017). A fast hybrid multi-site computation offloading for mobile cloud computing, *Journal of Network and Computer Applications* **80**: 219 – 231. JCR Impact Factor : 3.991 (2018).
URL: <http://www.sciencedirect.com/science/article/pii/S1084804516303514>
- Gu, F., Niu, J., Qi, Z. and Atiquzzaman, M. (2018). Partitioning and offloading in smart mobile devices for mobile cloud computing: State of the art and future directions, *Journal of Network and Computer Applications* **119**: 83 – 96. JCR Impact Factor : 3.991 (2018).
URL: <http://www.sciencedirect.com/science/article/pii/S1084804518302170>
- Jungum, N. V., Mohamudally, N. and Nissanke, N. (2020). Device selection decision making using multi-criteria for offloading application mobile codes, *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 326–331. Core Ranking : B (2018).
- Kwon, Y., Yi, H., Kwon, D., Yang, S., Cho, Y. and Paek, Y. (2016). Precise execution offloading for applications with dynamic behavior in mobile cloud computing, *Pervasive and Mobile Computing* **27**: 58 – 74. JCR Impact Factor : 2.974 (2018).
URL: <http://www.sciencedirect.com/science/article/pii/S1574119215001856>
- Lin, L., Liao, X., Jin, H. and Li, P. (2019). Computation offloading toward edge computing, *Proceedings of the IEEE* **107**(8): 1584–1607. JCR Impact Factor : 9.107 (2018).

- Lin, X., Jiang, J., Li, C. H. Y., Li, B. and Li, B. (2020). Circa: collaborative code offloading among multiple mobile devices, *Wireless Networks* **26**(2): 823–841. JCR Impact Factor : 1.981 (2018).
- Nguyen, Q.-H. and Dressler, F. (2020). A smartphone perspective on computation offloading—a survey, *Computer Communications* **159**: 133 – 154. JCR Impact Factor : 2.613 (2018).
URL: <http://www.sciencedirect.com/science/article/pii/S0140366419319401>
- Saha, S., Habib, M. A. and Razzaque, M. A. (2016). Compute intensive code offloading in mobile device cloud, *2016 IEEE Region 10 Conference (TENCON)*, Singapore, Singapore, pp. 436–440. Core Ranking : C (2018).
- Tao, X., Ota, K., Dong, M., Qi, H. and Li, K. (2017). Performance guaranteed computation offloading for mobile-edge cloud computing, *IEEE Wireless Communications Letters* **6**(6): 774–777. JCR Impact Factor : 3.096 (2018).
- Wang, S., Zhao, Y., Xu, J., Yuan, J. and Hsu, C.-H. (2019). Edge server placement in mobile edge computing, *Journal of Parallel and Distributed Computing* **127**: 160 – 168. JCR Impact Factor : 1.815 (2018).
URL: <http://www.sciencedirect.com/science/article/pii/S0743731518304398>
- Wu, S., Mei, C., Jin, H. and Wang, D. (2018). Android unikernel: Gearing mobile code offloading towards edge computing, *Future Generation Computer Systems* **86**: 694 – 703. JCR Impact Factor : 4.639 (2018).
URL: <http://www.sciencedirect.com/science/article/pii/S0167739X17329734>
- Zhao, X., Yang, K., Chen, Q., Peng, D., Jiang, H., Xu, X. and Shuang, X. (2019). Deep learning based mobile data offloading in mobile edge computing systems, *Future Generation Computer Systems* **99**: 346 – 355. JCR Impact Factor : 4.639 (2018).
URL: <http://www.sciencedirect.com/science/article/pii/S0167739X19304406>
- Zhou, B., Dastjerdi, A. V., Calheiros, R. N., Srirama, S. N. and Buyya, R. (2017). mcloud: A context-aware offloading framework for heterogeneous mobile cloud, *IEEE Transactions on Services Computing* **10**(5): 797–810. JCR Impact Factor : 4.418 (2018).