National College of Ireland

A Robust Computation Offloading Methodology for Improved Mobile Performance and Throughput

MSc Research Project
Cloud Computing

Swapnil S Vernekar
Student ID: 21174041

School of Computing
National College of Ireland

Supervisor: Yasantha Samarawickrama

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Swapnil S Vernekar
Student ID:	21174041
Programme:	MSc. in Cloud Computing
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Yasantha Samarawickrama
Submission Due Date:	14/08/2023
Project Title:	A Robust Computation Offloading Methodology for Improved Mobile Performance and Throughput
Word Count:	7591
Page Count:	22

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.

Signature:	Swapnil S Vernekar
Date:	17th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
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.	☐

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):	

A Robust Computation Offloading Methodology for Improved Mobile Performance and Throughput

Swapnil S Vernekar
21174041

Abstract

The main objective of offloading the task to a remote cloud is to reduce the battery power consumption, improving performance and life of the mobile device. Lots of work have already been done by the researchers in this domain, however there is still a considerable amount of improvement that can be achieved. This research will be focusing on reducing the time of execution and the CPU usage thereby improving the battery life of the device. A novel decision making algorithm named Mobile Offloading Computational Algorithm (MOCA) is been implemented which makes use of latency, bandwidth, live location of the device along with the previous historical records of that location while taking the decision. The algorithm makes use of a special performance equation which makes it possible to make accurate decisions. Using this algorithm, it has been seen that the execution time and the utilization of CPU has considerably reduced hence saving power and improving user experience.

1 Introduction

There have been a lot of advancements in the application development sector. These advancements brings multiple complex algorithms which require heavy computation power but the mobile devices which are used often does not have such powerful hardware which restricts the maximum throughput of the underlying software algorithms hence hindering the performance of the entire system. Due to this reason, computation offloading was brought forward. This involves offloading complete or certain part of the algorithm which potentially require heavy computation to the remote cloud server which processes it with high computation power hence results in saving battery power and improving performance of the entire system as a whole. A lot of research has been done in this domain and there is still a large scope for improvements in this sector.
The algorithms which help in offloading the part of heavy computation comprises of series of decision making steps which help in deciding whether the task needs to be offloaded or needs to be allowed to run on the local hardware. There are a loads of network parameters which are taken into consideration such as the bandwidth, network strength etc, however, there is very limited research involved on offloading systems which consider latency of the network in addition with the live location of the device on which the application is installed on. Involving latency as the parameter for the decision making process is very critical as considering only the bandwidth of the connected network cannot gurantee that the network can be trusted for the computation offloading process because even if the

network has high bandwidth with superior speed, the latency can be involved which retards the network speed from the target web address.

Research Question:- Considering the gaps in the ongoing research, following research questions can be taken into study:

1. To what extent CPU utilization and time of execution be improved by tracing the live location and the latency of the underlying network along with other network parameters for decision of offloading?

2. How can the historical records of offloading be used to optimize the decision making process of mobile offloading and computation algorithm (MOCA)?

The structure of the report is as follows:-
The Section-2 will give the detail analysis of the previous works done by the researchers in this domain. Section-3 will provide insights to the methodology on how this research is been carried out. Section-4 gives the design approach which is followed and the architecture which is used for designing this system. Section 5 will elaborate more on the implementation approaches, followed by sections 6 and 7 which gives a discussion of the research and the future work which can be done to improve the system. Finally Section 8 provides a list of references which are used to carry out this research work.

2 Related Work

Following are some of the main existing works which has been done in mobile cloud offloading domain which are as follows:

2.1 Previous works related to mobile cloud offloading

Chun et al. (2011) came up with a system named CloneCloud which is an offloading framework which makes use of threads for the process of computation offloading. The special part about this framework is that logic of the application is broken down into small minor parts or the modules and only small part of the application which can be offloaded is sent to the cloud for the process of remote computation. This entire thing is done using the two main modules of this framework which are mainly the static analysis and the dynamic profiling module. The use of threads helps this framework to execute and run the threads in concurrent manner such that some part of the code is run on the local device which is the mobile phone and other part of the code is executed on the cloud. Any dependency which arises on executing the code on the local is taken care by pausing its execution on the device until the computation results are obtained from the cloud for managing the dependency.

Kwon et al. (2016) researched on the performance metrics and came up with a performance predictor making use of f_Mantis which was designed to predict the performance in real time . This system accurately predicted whether the process of offloading will improve the performance of the system and it also provided the value of the real-time power consumed by system. In addition to this, he also proved empirically that the performance of the system actually increased in real time after making use of this predictor.

Kosta et al. (2012) introduced a framework named ThinkAir which proved that on employing this framework, the performance of the system increased drastically with comparatively less consumption of power. This was possible due to the virtual machine images which made the execution Parallelizable . Few of the simple benchmarking application algorithms along with certain complex applications were tested for the effectiveness of this framework. The framework comprised of many components and those components contained several modules. The component included the environment for execution of application, application server and the profilers. In addition to this, the profilers comprised of the hardware, software and lastly the network profilers. Using this framework they made a conclusion that the parallelizable application can execute over multiple virtual machine's.

Yet another framework was brought forward by Beloglazov and Buyya (2015) named MobiCOP-iot which made used of both, the cloud and the edge environment and evaluated that the performance increase was 9 times in cloud and 16 times in edge. They showed that the system had the capability to scale horizontally with varying loads without any loss in performance. The algorithm in the framework operated in 3 different modes mainly the Optimistic mode, Concurrent mode and the cloud-first mode. They tested this framework on four different algorithms which were the N-Queens, RenderScript, Video-processing and lastly the Chess problem. As stated above, this framework showed 9 times performance increase in cloud were as in the edge environment, it was 16 times increase in the performance.

Xia, Ding, Li, Kong, Yang and Ma (2014) introduced a system named Phone2Cloud which was focused on reducing the amount of energy that was consumed while the task was executed on the system. The logic behind its decision making algorithm was that it calculated the average time that it would take to execute the task on local device. It then fixed certain delay tolerance for the task and this tolerance is then compared with the average execution time. If it is found to be less than the threshold value then the task is offloaded to the cloud else, the power consumption of the local and cloud is estimated and is compared with each other, if energy consumption of the local is less than cloud then task is executed on local else it is offloaded on the cloud.

An algorithm named MAO(Mobile Application's Offloading) was introduced by Ellouze et al. (2015) which was invoked using two conditions- load on the CPU and the battery charge value. They used simulations which were numerical in nature in order to get the efficiency of the algorithm and also how it performs based on the jobs which are rejected by it and the energy which is saved. What this algorithm does is it breaks the task in small jobs and then it compares the delay in execution to that of delay which is set for that particular job and the state of charge. It checks if the battery charge is less than or equal to 20%. If yes then it checks if the task is energy efficient and if it is not then the job is rejected else it checks if it meets the quality of service. If all these conditions are met then the job is offloaded else next job is taken.

2.2 Previous works related to mobile cloud offloading taking location into consideration

Neto et al. (2016) came up with a decision making engine which used location as one of its parameters for the process of offloading. This is a hybrid engine which can be used in any of the offloading framework. After the evaluation, it was found that there was about 50% less energy consumption by the hardware while computing and there was around 10% less CPU overhead. In this algorithm, certain number of annotations were used in order to identify what part of the code needs to be offloaded and what all parts of the code needs to be run on the local device itself. This system comprises of three main components which are Instrumentation module, Decision making engine and the connector component. The instrumentation module is responsible for fetching the parts of the application logic which contains the annotation and then transfers it to the decision making engine for further process. The DME (Decision Making Engine) on the other hand decides whether the task can be offloaded or not. To make this decision, it considers the amount of CPU that will be used and the battery energy that will be consumed if it was executed on the local device. Based on this appropriate decision is made. Once the task is offloaded to the cloud, the connector module comes into action which is responsible for remote connection of the cloud environment to the local. It also serializes and deserializes the objects as and when needed. Furthermore, this engine makes use of location as one of its main parameters and based on the traced location of the device, the bandwidth of the connected network is calculated and inorder to do so, Shannon's equation is used which internally makes use of SNR value to get the final bandwidth.

Xia, Liang, Xu and Zhou (2014) introduced an online location based decision making system in which the units were arranged in two tiers, the first one being the cloudlets and the second being the remote clouds which aimed to maintain their SLA while using same amount of energy. This system makes use of location as one of its parameters for offloading. Each Access Point in the particular location can have is own local cloudlets and the tasks that needs to be executed might be executed on the local device, local cloudlets or over the remote could environment. Each location can have a lot of workload at any point of time over its local cloudlet hence it would be reasonable to execute that task over the local device to save the time else it can be offloaded on the cloudlet or the remote cloud for its execution and computation if there is not much workload. This is where the location as a parameter is considered. However, it is not actively in use to make the decision to offload the task.

2.3 Other related work relating to mobile cloud offloading

Boukerche et al. (2019) made a comparison of the various algorithms which were used for the purpose of computation offloading process. It compares all the architectures of the systems and also provides the advantages and the disadvantages of the same. In addition to this, it also provides the open challenges and the future directions which can help to uncover this research area more deeply.

Karunanithi (2020) designed a system and a smart algorithm which had two main use cases, first one being the QR-Code scanner and the image processing. QR-Code scanning was considered as a latency intensive iteration where as image processing was considered

as a computational intensive task. The cloud platform that was used is Amazon's AWS and the application was developed using Android Studio Code. The algorithm that was implemented had the ability to handle both, computational intensive task as well as the Latency intensive task. Normal device parameters were used in this algorithm such as the battery level, free memory space and the connectivity state of the mobile device. The future work of this paper was to implement a machine learning algorithm. The implementation in this particular paper is the continuation of the research that was done by Karunanithi (2020) however, instead of machine learning algorithm, a smart algorithm will be used which will take into consideration the previous decisions that were made along with the bandwidth, latency and the location of the device in order to improve the decision making process. The base of this research will be the same however, the algorithm that was used will be completely changed to improve the time taken to complete the task.

3 Methodology

This section will elaborate on the methods which are used in order to make this research successful along with the tools and the technologies used to achieve the results. The task, based on the evaluation by the inbuilt custom algorithm developed the decision is made whether to run the task on the local device i.e mobile phone or to offload and execute the task on the cloud and get the results to be displayed on the mobile screen for further studies and conclusion.

3.1 Tools, Technologies and Techniques for application development

The base application is installed on the android emulator to be better connected with the underlying application code, however this application APK can easily be installed on the mobile device and can be tested out on the same. For the purpose of development, Android Studio Code is used which is 2022.2.1 Patch 2 Flamingo version.
The Virtual Machine embedded on this studio is of OpenJDK which is a 64-bit server Virtual Machine. Furthermore, Pixel 6 Pro is been used as an Emulator of the device on which the application will be installed and run upon. This virtual device is running on Android API 34 Google APIs. The entire application is developed using JAVA programming language of version 8. Several Async functions are used in order to allow the application to run smoothly while performing background activities.

3.2 Application algorithm used for the purpose of system testing

The application which is developed on android studio is embedded with an image processing algorithm used to transform the image captured through device camera. Image processing is a very high computing task and requires potentially higher computation power than any other activities. It requires a use of Graphical Processing Unit (GPU) to manipulate and render each of the tiny pixel in the captured image. Each of the image captured are a very large data composed of high number of small pixels which are divided into RGB which is basically a combination of 3 different colors RED, GREEN

and BLUE. Higher the resolution of the camera, higher is the size of the image to be processed as it will contain a lot of small individual pixels needed to be rendered which is computationally intensive task hence requiring more CPU power.
The more computation the CPU does, more is the battery consumption by the device hence an efficient decision making algorithm needs to be in place so that the task is offloaded on the remote processing area in order to save some battery of the local device is low on power. The decision made by the offloading algorithm is based on various different variables and amongst them, bandwidth and the latency along with the location plays a vital role in predicting the better execution environment (local or the cloud). If the decision is made to be offloaded on the cloud environment, then the API call is done by the algorithm which invokes the algorithm which is present on the cloud server. This API call is possible using AJAX technique. After the processing, the results are then sent back to the device in the form of JSON which are then extracted from the template to be rendered and displayed on the device screen.

3.3 Performance Equation

The decision making algorithm which is the main module for the process of deciding the execution environment relies on the performance equation along with other parameters in order to make the decision. This equation consumes values which are pre-fetched before algorithm is executed. These parameters include weights W1,W2,W3 and W4, the bandwidth, latency, signal strength and lastly the current battery level of the device. The values of the weights which are specified above depends on the priority of the algorithm to select amongst other parameters which are specified above. There is a threshold value of the performance which is set below which it will take the decision to execute the algorithm on local device. If the live performance value which is calculated is more than the threshold then the decision to offload the task to the cloud will be taken.

3.4 Offloading task to the cloud server

The algorithm present on the local device is replicated on the AWS cloud services. Microservices architecture is used in order to achieve the results. The main application logic is written using Node.Js and then all the node modules are installed. As the application size after installing all the node modules is greater than 10MB, it is first uploaded on S3 and then the zip file for the same is pulled by the lambda service. Lambda is very light weight execution environment and is serverless. All the code present on the serverless is in the form of functions which are invoked as and when needed. Apart from this, it is highly scalable which gives it the ability to expand or retract depending on the application load. Due to the above mentioned specifications lambda was found to be the best environment for execution. The API gateway is connected to the lambda function which acts as a middleware to route all the requests to their respective lambda endpoint.

The monitoring of the cloud server is done using cloud watch service of AWS which gives detailed logs of the services which are running. Using this any issues pertaining to the lambda can be identified and be resolved.

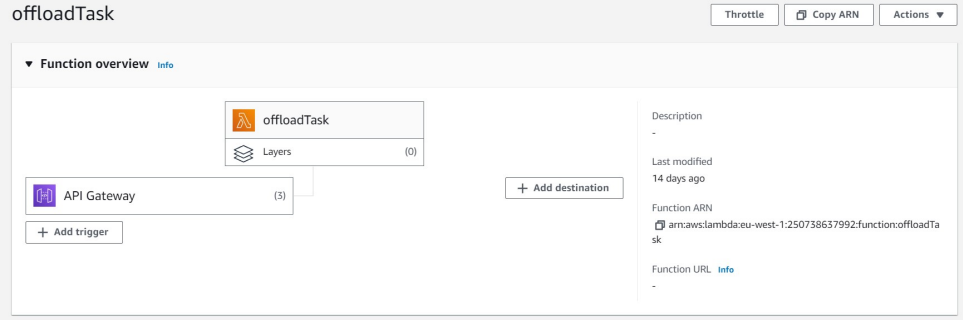

Figure 1: Cloud Server Architecture using API Gateway and Lambda

4 Design Specification

Before any task is offloaded on the cloud server, lots of background parameters are checked to make sure that the decision made is precise and accurate when it comes to mobile performance. For this purpose several modules are designed each having specific evaluation function which scans the underlying service and helps the engine to make the appropriate decision on whether the task needs to be offloaded or run on the local device.

For the purpose of this research four main components are developed and used which are namely the Device Analyzer module, Network Analyzer module and the Location detector. The fourth one is the performance calculator which calculates and provides with an integer number based on the output of other modules.

4.1 Architecture diagram of the system

Figure 4 below shows the architecture of the entire system under research with each of its modules along with the control flow. The Network Analyzer, Device Analyzer, location detector along with the decision making engine and performance calculator are the part of Mobile Offloading and Computational Algorithm (MOCA) .

4.2 Device Analyzer

This is the module designed to fetch all the real time data about the android device on which the application is installed on. Android studio code, the IDE on which the application is developed comes with a set of plugins which help in getting a lot of hardware specific information about the device. For the implementation of this algorithm in this research, memory information is collected using the activity manager module present on android studio which provides with all the related information of the device memory. Apart from the memory, battery level and the state of the battery is collected using the battery manager module. All these parameters is stored into a hash map in the form of key and value pair which is fetched by other modules for further processing.

4.3 Network Analyzer

The process of offloading is entirely dependent on the values obtained by the network analyzer module. This component fetched important network information and provides

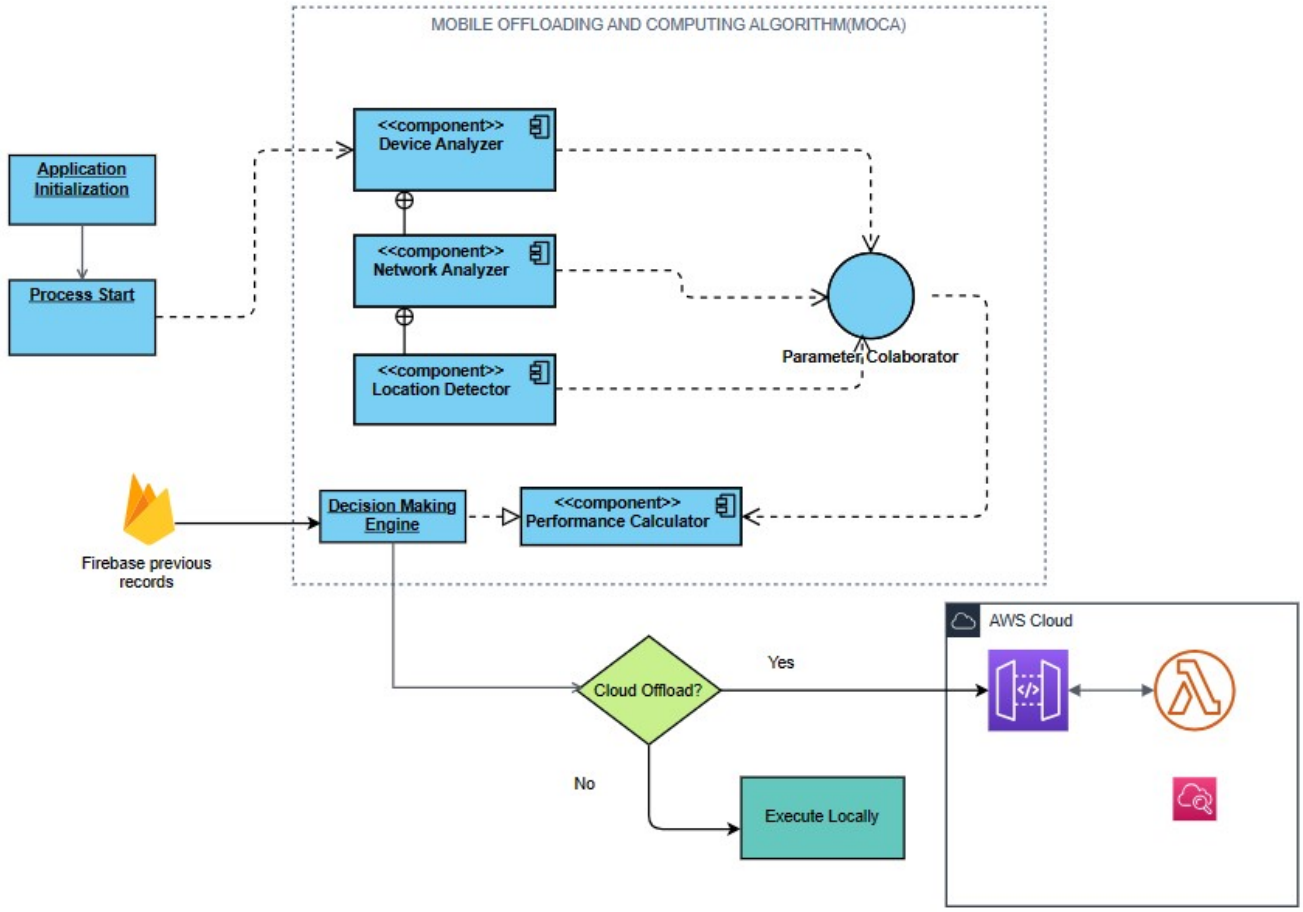

Figure 2: Architectural Diagram of the system

it to the decision making engine for analyzing the values. This checks whether the mobile device is connected to the internet and if so, then it check what type of data networks it is connected to. The wifi and the mobile data can be two types of data network that the device can be connected. The android.net.ConnectivityManager and the android.net.NetworkInfo are the two main device packages which this component makes use of. All the values of the network obtained is then stored into a hashmap so that they can be used to make appropriate decision by calculating the performance which acts as a main parameter for making the offloading decision.

This module is also responsible for checking the latency of the connected network. Getting the actual network latency is very challenging as it requires a lot of background activity while the application is running. Implementing it with the normal approach cannot be possible to get the precise value hence, async task implementation was followed. This is a kind of programming approach followed when a certain important task needs to be carried out while the application is still under processing state. This is quite a complex approach but it allows smooth process of the entire task under execution.

4.4 Location Detector

This algorithm makes an active use of live location of the device while the application is executed along with the bandwidth and latency of the network. These parameters play an important role in the performance calculating module in order to make an appropriate decision of offloading a task. The output of this module is stored in a data store and is used actively by the algorithm for improving the time of execution every time the application is opened at the same location. The android package named Location manager is used to get the precise latitude and longitude of the device. The LocationManager.GPSPROVIDER is the plugin used to fetch the location by making use of GPS actively.

4.5 Performance Calculator

This component is responsible for calculating the performance based on the values provided by the three analyzer modules. It consumes the bandwidth, latency, signal strength and the battery level values stored into the hashmap. Based on these values the estimated performance is calculated and is provided to the decision making engine for further analysis. The performance equation used is as follows:-

$Performance = W1 * bandwidth + W2 * (latency^{-}1) + W3 * signalStrength + W4 * batteryLevel$

Where W1, W2,W3 and W4 are the weights which can be adjusted.

For this research work, latency is the most important parameter followed by bandwidth hence W2 is given the value 1 followed by bandwidth as 0.3 and W3 and W4 are given the values 0.2 and 0.1 respectively. This will be termed as Equation-1 and will be referred throughout this paper.

4.6 Firebase Plugin

The android application is connected with the firebase plugin which provides a lot of additional features. It is a service provided by google which is connected to the google cloud service. For this research it is used for the purpose of data storage medium and performance tracking interface. There are two tables which are maintained for storing the values traced out by the algorithm which are offload_records and history table. The first table which is the offload_records table maintains every record of each decision made by the algorithm. Its field include the latitude, longitude, execution environment and the total time taken for the execution. This table is made to visualize the time taken by the algorithm to execute on the two platforms, cloud and the local. The second table is the history table which maintains a single record per location. It updates the value of its field when the performance is changed or when the algorithm detects that its the best location for offloading for future executions. Its field include bandwidth, best location, latitude, longitude, performance and the execution environment information. Apart from the firebase database, it also provides insights on total CPU that was utilized while executing the application along with the graphical representation of change in application response time.

4.7 Decision Making Engine

This module is responsible for making appropriate decisions based on the performance value provided by the performance calculator and the previous values obtained by the firebase database for that particular location. It makes the decision to either execute the task on local device or the cloud environment.

4.8 Cloud Offloading

If the performance value obtained from the performance calculator is greater than the threshold value then the task is offloaded to the cloud environment. The architecture followed on the cloud is microservice type where the main application logic is deployed on the lambda service which is in the form of function which gets invoked as and when API gateway is hit by the appropriate request. For this research, post request is used and the result is obtained in the form of 64 bit image which is decoded on the local device after the cloud response is successful and the result is displayed on the device screen. For monitoring purposes, cloud watch service of AWS is used which provides cloud logs of in very detail which can be used for resolving any cloud related issues which may arise.

5 Implementation

5.1 Mobile Offloading Computational Algorithm

For making this offloading system a custom algorithm named Mobile Offloading Computational Algorithm is designed which is shown below in two sub parts. The first part shows the basic structure of the algorithm where as the second one shows the actual breakdown of the internal logic which is used by the system for making the decision of cloud offloading.

Algorithm 1 Mobile Offloading Computational Algorithm (MOCA)-1

```
0: if NetworkConnectivity then
0:   if Charging == True then
0:     if avaiMemPct ≤ 50 then
0:       makeDecision()
0:     else
0:       executeOnLocal()
0:     end if
0:   else
0:     if avaiMemPct ≤ 50 or batteryPercentage ≤ 20 then
0:       makeDecision()
0:     else
0:       executeOnLocal()
0:     end if
0:   end if
0: else
0:   executeOnLocal()
0: end if=0
```

The algorithm first checks whether network connectivity is present on the device or whether it is disconnected from the network. If it detects the connection with the network provider then the algorithm proceeds with checking the charging status of the battery. If the device is connected to the power then next step is to check the memory available. The algorithm is designed in such a way that only when the memory available on the device is less than 50% only then it will check for further conditions for offloading. Free memory availability in the device ensures that whenever a task is running on the local, it will have space when the application data increases which can add up some level to performance, however when there is less or no free space available on the device, the performance can decrease which can hinder the user experience hence it becomes a right opportunity to offload while checking with other parameters which is done by the second part of the algorithm.
The state of charge detection is added in this approach as it will ensure that the device has adequate power supply from either of the power sources i.e the charging device or the battery. If the algorithm detects that the device is disconnected from the power supply and is dependent on the internal battery then it will check for the amount of charge that the battery has at that unit of time. If the memory is less than 50% and battery percentage is less than the threshold which in this case is set to 20% then it will consider other conditions to make sure that the offloading step is the right decision and will not decrease the performance further. If any of the condition is not met, decision to run the task on the local device is taken and the task is run on the mobile device itself.

The second part of the algorithm shows the actual steps and parameters considered while making up appropriate and accurate decisions which helps in improving performance of the mobile device by reducing the execution time. After a call to makeDecision function is made, it calculates and fetches the latency of the connected network and for this, a ping to host name www.example.com is done which is a testing server made available to the developers all around the world. After getting the appropriate value of

Algorithm 2 Mobile Offloading Computational Algorithm (MOCA)-2 (MakeDecision())

```
0: location ← getLocation
0: latency ← getNetworkLatency
0: performance ← getPerformance
0: firebaseRecords ← getFirebaseRecords
0: if firebaseRecords.count == 0 then
0:    if performance≥ threshold then
0:       firebaseRecords.Store("state ← cloud")
0:       executeOnCloud()
0:    else
0:       firebaseRecords.Store("state ← local")
0:       executeOnLocal()
0:    end if
0: else if firebaseRecords.count ≥ 1 and firebaseRecords.env==cloud then
0:    averagePerformance ← getAveragePerformance()
0:    if averagePerformance ≥ threshold then
0:       firebaseRecords.bestLocation=true
0:       executeOnCloud()
0:    else
0:       executeOnLocal()
0:    end if
0: else if firebaseRecords.count ≥ 1 and firebaseRecords.env==local then
0:    if performance ≥ firebaseRecords.lastPerformance then
0:       averagePerformance ← getAveragePerformance()
0:       if averagePerformance ≥ threshold then
0:          firebaseRecords.bestLocation=true
0:          firebaseRecords.state=cloud
0:          executeOnCloud()
0:       else
0:          firebaseRecords.lastPerformance=performance
0:          executeOnLocal()
0:       end if
0:    else
0:       executeOnLocal()
0:    end if
0: else if firebaseRecords.bestLocation==true then
0:    executeOnCloud()
0: else
0:    executeOnLocal()
0: end if
=0
```

latency, location of device is detected using a plugin which makes use of GPS to get the live location of the device. The device, network and location analyzer is responsible for obtaining the values. All these values are stored into a hash map which links the value to its key. Next, the battery level which was detected along with the location and the latency of network is given for calculating the performance using the Equation-1 mentioned in this paper above. The weights W1, W2, W3 and W4 is adjusted as per the requirements, however, for this implementation W1 is given a value of 0.3 followed by W2 as 1, W3 as 0.2 and lastly W4 as 0.1 respectively. The reason behind W2 taken as 1 is that in this research, latency is given the highest priority to study the use of latency to improve the performance.

Performance Parameter weight	Value
Bandwidth W1	0.3
Latency W2	1
Signal Strength W3	0.2
Battery Level W4	0.1

After the performance is calculated, the call to the firebase is made to check for previous records. If no records exists then it check the performance value with the threshold value which in this research is taken as 8. If the performance is greater or equal to this value then the data is stored on the firebase and cloud execution is done. If the performance is less than the threshold then data is stored on firebase and local execution is done.

If there exist previous records, and is the previous record environment is cloud then the average performance is calculated using the current performance and previous performance value and this average performance value is checked if it is greater than or equal to the threshold, if yes then best location is set to true and is executed on cloud else it is executed on local.

If the previous record environment is local then it checks if the performance calculated is greater than or equal to previous performance value is yes then average performance is calculated and same approach is followed as explained above. The state of previous record is set to cloud and cloud execution is done. If the performance calculated is not greater than last performance value then the last performance value of firebase is updated with the latest performance value and the local execution is carried out. If none of the above conditions are met and if the previous record location is set to true then cloud execution is done else the task is executed on local. Figure 5 shows the flowchart of the algorithm.

5.2 Cloud Execution Environment

When the decision of offloading is made, the cloud execution starts. For the purpose of developing the cloud architecture, two of the Amazon's main services are used which are mainly the API gateway and the Lambda functions.

5.2.1 The API Gateway

The API is a service provided by Amazon which allows to develop several API based on Restful web services. These API can be connected to Lambda function so that it gets

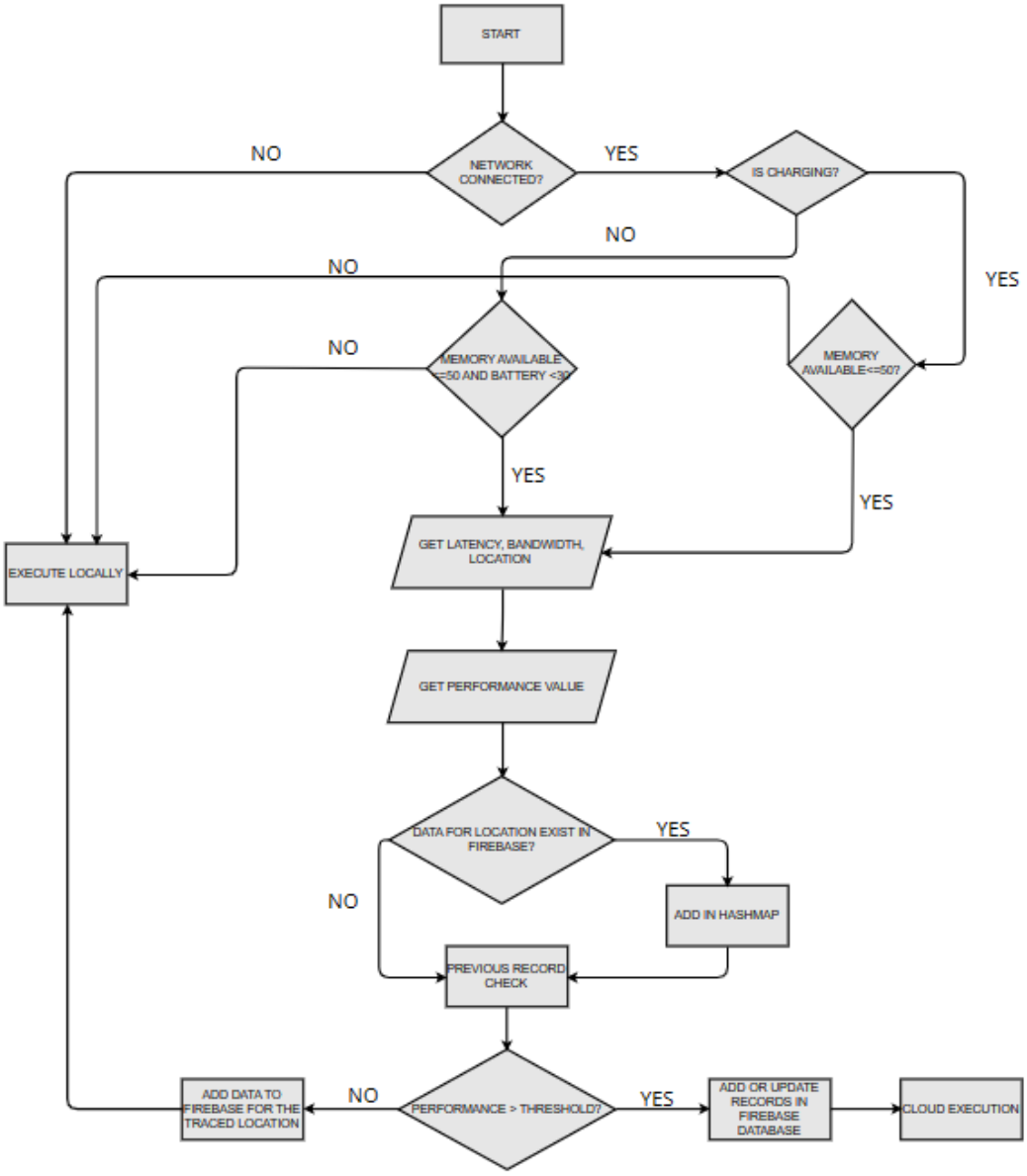

Figure 3: Architectural Diagram of the system

invoked everytime an API call is made. For the research purpose, an API is developed with POST request and this API gateway is connected to the Lambda function. Figure 6 shows the developed API gateway which is used for this research purpose. This API is tested on Postman before it is integrated in this system.

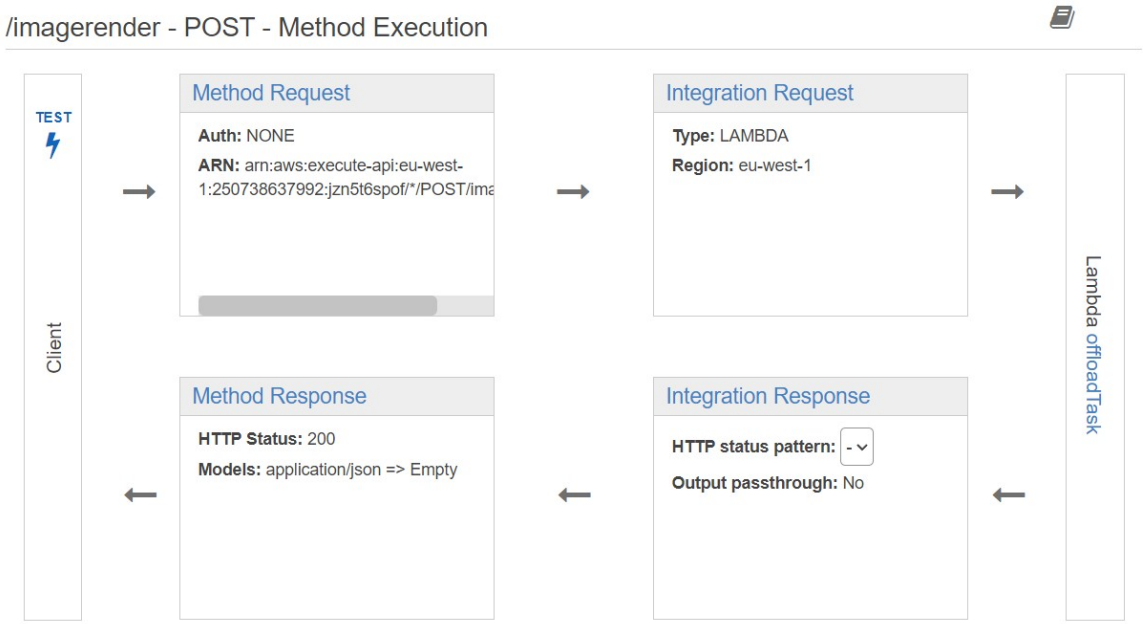

Figure 4: API gateway implementation

5.2.2 The Lambda Function

The entire image rendering logic in the cloud is been implemented on the Lambda service of AWS. Using Lambda promotes the microservice approach which makes the system loosely coupled hence allowing changes to the source code in future. Lambda follows a FAAS architecture where each implementation is in the form of a function. This can be invoked using API gateway and the response from this can be sent back. For the implementation process Node js is used and the code is first added to s3 bucket. Lambda function then pulls the code form this bucket and then is used as the API end function during offloading. It offers Cloud watch service out off the box which can be used for monitoring process of the application on the cloud.

5.3 Firebase Environment

All the decisions made by the decision engine is stored into the Firebase Datastore, For the purpose of the research, two tables have been maintained named History_Table and the Offload_Records. The first table maintains single record per location. This table includes Bandwidth, Best_location, Latitude, Longitude, performance and the store parameters. The decision making engine uses Best_Location and the performance field in order to make the decision based on the previous records. The Offload_Records on the other hand keeps track of each and every decision made by the system for the purpose of evaluation.

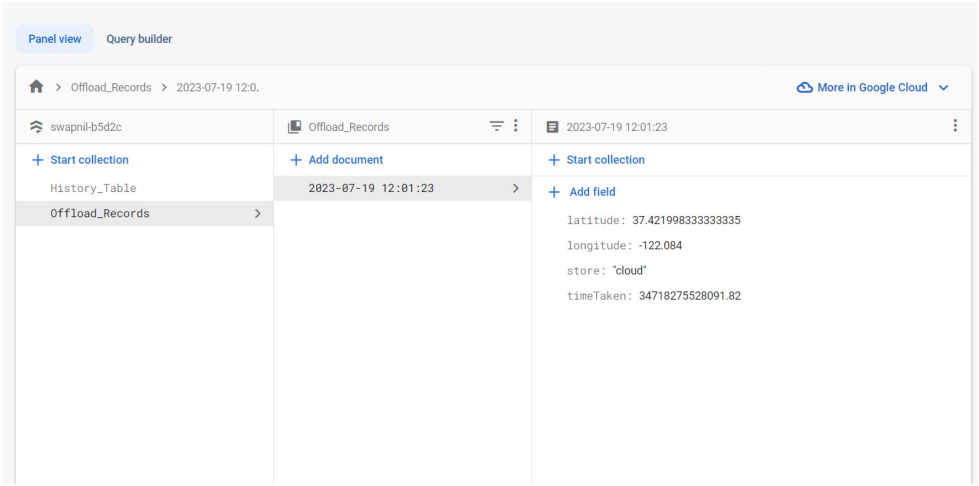

Figure 5: Firebase Records

6 Evaluation

For the purpose of evaluation of the proposed system, Firebase plugin is been integrated into the android studio code and the apk of the ready system is been deployed on the evaluation module to get the cpu usage. The pluggin is added to the android manifest.xml file and the grade build is invoked manually to pull all the related dependencies from the remote repository of Google. For the experimentation purposes, Oppo Realme X2 pro emulated device is used with API version 34 and having 2048 MB as the RAM memory. The effectiveness of the system is evaluated using multiple parameters such as the performance equation, time for execution on cloud vs local, CPU utilization, memory usage and also the platform selection. Each Experiments below are linked to each other and they together constitute the research findings.

6.1 Testing the Performance Equation of the System

The effectiveness of performance equation is done using 6 sub-epochs. First and the second epoch is grouped under Test-1, third and the fourth is grouped under Test-2 followed by fifth and sixth test under Test-3. As shown in the graphical representation, dark blue color indicates 5G/ WiFi Network connectivity whereas light blue shows that the device is connected to 3G/4G Network.
Test-1 was carried out at 11 am and was observed that when the device is connected to WIFI or the 5G networks, the performance came out to be around 26 where as the same performance equation gave a value of 5 when the network connectivity was 3G/4G network. On performing Test-2 at 6pm the same day, it was observed that there was not much difference in the values for these networks as compared to the results given by Test-1, however, 3G/4G network connectivity gave much lower value as compared to Wifi/5G network. Test-3 on the other end was carried out at night resulting in low latency on Wifi and 5G networks, hence giving much higher value of 60 while connected to 5G/ Wifi network connectivity. The reason behind choosing different time for evaluation of performance was that latency of the network decreases at night when compared to day as there is less network congestion and traffic hence proving that the parameters and weights chosen in the equation is valid.

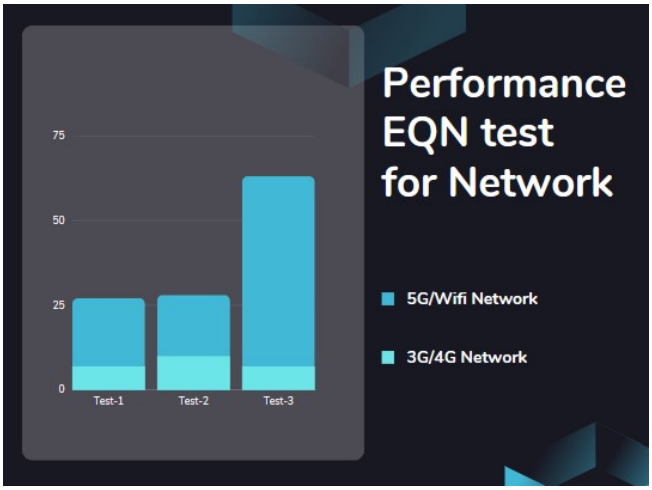

Figure 6: Performance Equation Output

6.2 Evaluation of time taken by the system on the two platforms

The aim of this research is to make an attempt to decrease the computation time by offloading the task to the cloud. The algorithm was run several times and the data on the firestore table is taken for plotting the time taken by the system to execute the computation on cloud to that on local. Following are the results that were obtained.

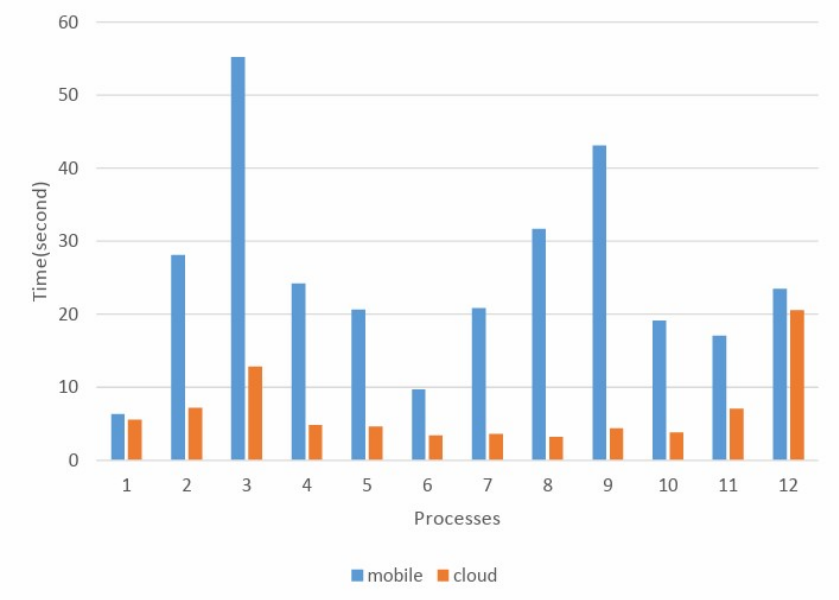

Figure 7: Time evaluation of the algorithm

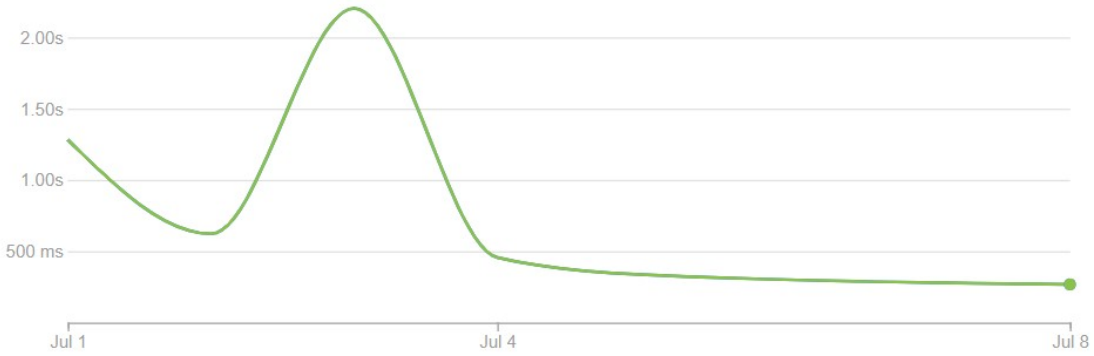

Figure 8: Time graph from Firebase performance evaluation module

A total of 24 epochs on the system were run and alternate epochs were grouped together as one single unit hence in the above bar chart, the x-axis of the system has 12 process where each process has two sub process one for the local and the other one for the cloud execution. It was studied that the time taken by the mobile execution is much more than the time taken by the cloud. For the process numbered 1 and 12, mobile network was switched from 3G to 4G hence there is very small difference between the time taken but it can be concluded that execution on the cloud is much faster when it comes to the time taken for the execution when compared with local device execution.

6.3 CPU and Memory Utilization on execution on Cloud and Local

The network was constantly switched between the mobile and the Wifi and the algorithm was run for several iterations to study the CPU and memory utilization in this process. Figure 9 and 10 shows the CPU utilization obtained from firebase console.

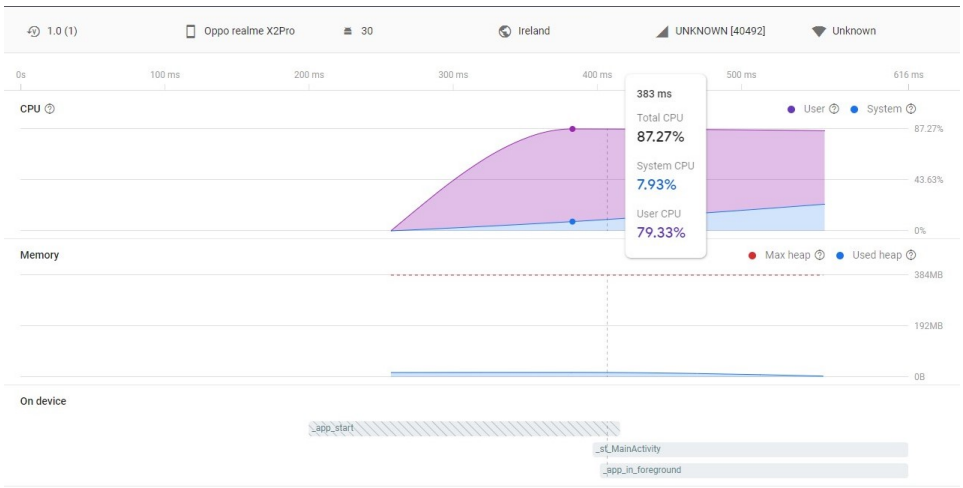

Figure 9: CPU utilization on local execution

The mobile data was turned off and the network was disconnected from the provider, The image editing application was then run. After the the execution was completed, firebase analyzed the CPU utilization and the above result was obtained. It was seen that the total CPU utilization was 87.27% which included the CPU utilization and User CPU utilization which was 7.93% and 79.33% respectively.

After the mobile execution was completed, the network was turned on and the device was connected to WiFi network and the application was re-executed. After completion of execution, it was observed that the total CPU utilization was 10% which was 8 times less than the CPU utilized during local execution. Furthermore, when the memory was analyzed, it was seen that the utilization was quiet less than local execution hence proving that offloading is reducing the utilization of CPU.

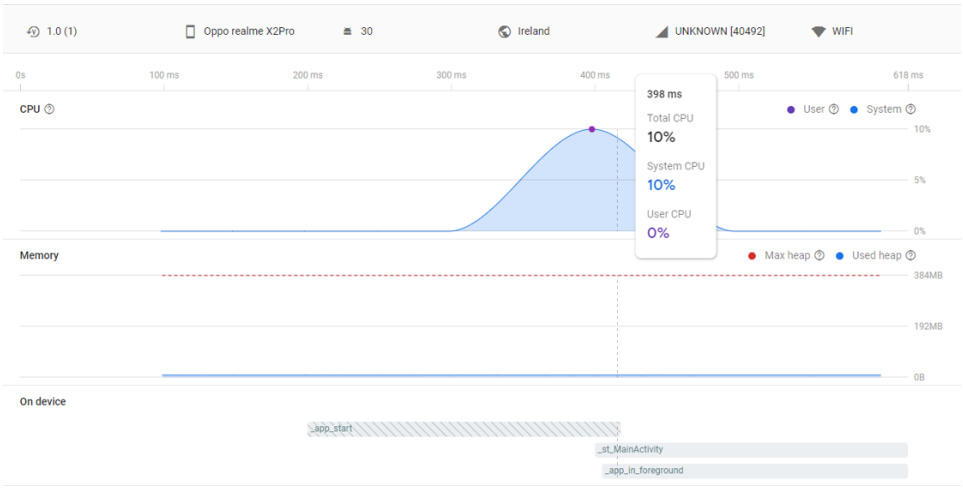

Figure 10: CPU utilization on cloud execution

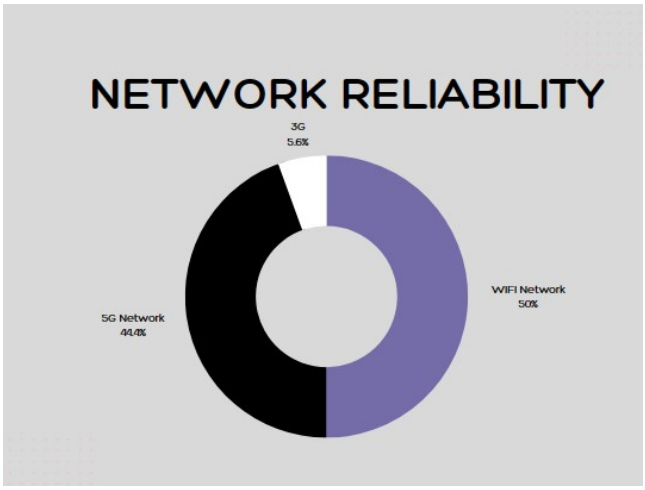

Figure 11: Network Reliability

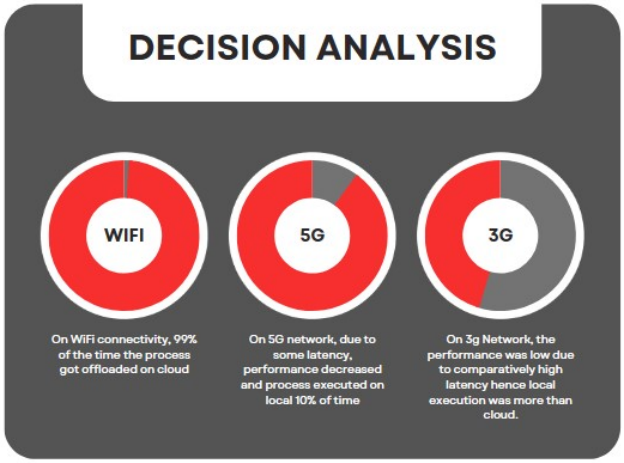

Figure 12: Analysis of the decision made

6.4 Discussion

This Mobile Offloading Computational Algorithm designed and implemented shows that the time taken for completing the execution is considerably reduced. Apart from this, the app startup time reduced from 2 seconds to less than 500 milliseconds. The algorithm implemented by Karunanithi (2020) didn't took into consideration the latency and the previous offloading records. The implementation approach followed in this research took into consideration latency of the network and also the previous records of offloading hence making it reliable system improving CPU performance. The processor utilization was considerably less hence reducing battery consumption. Few network conclusions are obtained and are shown in the figure 13 and 14 respectively.

Figure 11 shows that the most reliable network which was evaluated based on the decision taken to improve the performance of the device was the WiFi network followed by the 5G network with 50% and 44.4% reliability respectively. The least reliable network for offloading was the 3G network which was only 5.6% reliable.

Figure 12 demonstrates the analysis of the decision made on different networks. It was seen that when the device was connected to WiFi network, the latency was less and it had high bandwidth hence performance value was more than the threshold which resulted in the task getting offloaded on the cloud 99% of the time. On 5G network, the latency was comparatively more than WiFi hence 10% of the execution was on local where as rest 90% of the time the execution was on cloud. On the other hand, when 3G network was used, the performance value obtained form the equation was considerably less due to high latency which resulted in executing the task on local maximum times than on the cloud platform.

7 Conclusion and Future Work

The objective of this research work was to make use of latency of the network along with bandwidth and live location of the device to improve the performance of the system as a

whole. It made use of a custom made algorithm named Mobile Offloading Computational Algorithm which had a special performance equation consisting of the above mentioned parameters to calculate a performance value based on which appropriate decision was made while taking the historical values into consideration. This approach was continuation of the existing work done by Karunanithi (2020) with considerable modification in the algorithm already implemented. On evaluation, it was found that the performance was increased to an extent by reducing the CPU utilization 8 times while executing the task on the cloud environment. Adding further, it also demonstrated how the time of execution was reduced when the task was offloaded.

This research didn't took into consideration then edge devices which can be used when there is no network connections. Incorporating edge devices to this system can be a part of future work along with implementation of a trained machine learning model which can smartly decide the execution environment.

References

Beloglazov, A. and Buyya, R. (2015). Openstack neat: a framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds, *Concurrency and Computation: Practice and Experience* **27**(5): 1310–1333.

Boukerche, A., Guan, S. and Grande, R. E. D. (2019). Sustainable offloading in mobile cloud computing: Algorithmic design and implementation, *ACM Comput. Surv.* **52**(1).
URL: *https://doi.org/10.1145/3286688*

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M. and Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud, *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, Association for Computing Machinery, New York, NY, USA, p. 301–314.
URL: *https://doi.org/10.1145/1966445.1966473*

Ellouze, A., Gagnaire, M. and Haddad, A. (2015). A mobile application offloading algorithm for mobile cloud computing, *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pp. 34–40.

Karunanithi, N. e. a. (2020). Mobile offloading technique for latency-sensitive and computational-intensive task, *2020 Masters thesis, Dublin, National College of Ireland.*

Kosta, S., Aucinas, A., Hui, P., Mortier, R. and Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, *2012 Proceedings IEEE INFOCOM*, pp. 945–953.

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.
URL: *https://www.sciencedirect.com/science/article/pii/S1574119215001856*

Neto, J. L. D., Macedo, D. F. and Nogueira, J. M. S. (2016). Location aware decision engine to offload mobile computation to the cloud, *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium* pp. 543–549.

Xia, F., Ding, F., Li, J., Kong, X., Yang, L. and Ma, J. (2014). Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing, *Information Systems Frontiers* **16**.

Xia, Q., Liang, W., Xu, Z. and Zhou, B. (2014). Online algorithms for location-aware task offloading in two-tiered mobile cloud environments, *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, UCC '14, IEEE Computer Society, USA, p. 109–116.
URL: *https://doi.org/10.1109/UCC.2014.19*