DME: Technique for computation offloading in Mobile Cloud Computing.

MSc Research Project
Master of Science in Cloud Computing

Muhammad Abu Bakar Sani
Student ID: 17112044

School of Computing
National College of Ireland

Supervisor: Shivani Jaswal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Muhammad Abu Bakar Sani
Student ID:	17112044
Programme:	Master of Science in Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Shivani Jaswal
Submission Due Date:	19/09/2022
Project Title:	DME: Technique for computation offloading in Mobile Cloud Computing.
Word Count:	5715
Page Count:	19

Signature:	Muhammad Abu Bakar Sani
Date:	19th September 2022

DME: Technique for computation offloading in Mobile Cloud Computing.

Muhammad Abu Bakar Sani
17112044

Abstract

Around the world, the advancement and utilization of mobile devices have been increasing tremendously. In spite of the improvement and evolution of these devices, they are still considered as resource constrained devices because of its portability. The advancement in mobile device technology results in enabling them to execute complex applications and/or programs seamlessly but consumers are becoming more demanding and expecting the battery to last longer along with execution of multiple intensive tasks. Hence, Mobile Cloud Computing (MCC) plays an important role in order to enhance the capabilities of devices using offloading methods. In this paper, a novel algorithm has been proposed called Decision Making Engine (DME) which will be executed on a mobile device to give suggestion whether to offload task to the cloud or to run it locally on a device. It evaluates various parameters such as battery information, hardware specifications (Memory usage) and network accessibility before providing any suggestion. Task offloading has ability to free some memory and to extends battery life which improves the device performance. An android application and a microservice with Fibonacci series are developed to evaluate the experiment. The experimental results show that the DME works as expected and based on the execution time in the two environments, the execution time significantly reduces when performing execution on the cloud than on the local device. Cloud execution also free-up device resources for other tasks.

1 Introduction

There has been a remarkable increase over the last few years in the usage of mobile devices as with the advancements in its hardware specifications. The mobile devices include smartphones, wearable, tablets etc. The applications able to perform complicated tasks are also rising and getting more complexed. Video Streaming, Live Gaming and Face Recognition are some of the examples of such applications. These types of applications use various mobile resources i.e., CPU, memory, battery usage etc at a higher rate. The benefits of mobile devices include portability, easily accessible and affordable etc. Many of the mobile devices do not last for a day due to highly intensive use of device's CPU. In such cases MCC facilitate users by improving the execution and battery of mobile devices. It integrates network, mobile, and cloud computing.

MCC brings the massive quantity of cloud resources to strengthen the resource constrained devices. Google Cloud, Amazon Web Services, Microsoft Azure etc are some of the CSPs (Cloud Service Providers) which provide resources that are straightforward and easy to integrate such as storage, virtual machines etc. They work on the Pay-As-You-Use model. These resources can be quickly provisioned and merged to be used by the functions running on the devices.

MCC has been proposed mainly for resource constrained devices to overcome number of issues. The key concept of MCC is to offload resource intensive tasks from device to the cloud. The cloud execution of these tasks is better than local [1]. The very important part in offloading is to make appropriate decision. The decision-making algorithm is used within the application which will decide during run-time whether to offload task or not [2]. There are number of decision-making algorithms that have already been developed which are only be able to handle particular sort of task.

Along with huge benefits, MCC has some of the limitations like low flexibility, high power consumption, singular offloading selection, huge costs etc [2], [3]. These limitations need to be considered and tackled before taking up MCC. As mentioned by the authors [1], [4], MCC is made of two models i.e., task delegation and code offloading. The code offloading is associated with designing the application which includes to get resource exhaustive tasks and then offload them to external platforms. The task delegation is, in which device uses cloud services and announces possible interoperability issues. Offloading is the feature around which this project revolves. Mobile code offloading and Computational offloading are the two terms that can be utilized interchangeably. In this, code is usually partitioned and profiled with set goals in mind. These goals involve decrease in power consumption, improve in performance and scalability.

Despite of the detection of the intensive tasks and involvement of decision algorithm to determine whether to offload a task or not, a problem is still around to accurately perform offloading of compute intensive task [4]. It is important to confront this issue as incorrect profiling and partitioning can damage the mobile device. As authors mentioned [5], partitioning can either be dynamic or static with the difference between them is their execution timing like dynamic partitioning occurs during application run-time

Research Question: Can the application makespan for computational intensive task offloading applications be improved using a novel decision-making engine based on current context of mobile device such as Network, Battery and Memory?

In this research project report, an algorithm has been proposed naming Decision-Making Engine (DME) which works on dynamic partitioning. The proposed algorithm considers various parameters such as connection status, battery level, battery health, battery status, and available memory, which further helps in making a decision regarding sending task to the cloud or to the local server. The DME works on smartphone devices.

The rest of this document is split into various sections and subsections. Section 2 illustrates the related work in this area, Section 3 introduces the employed methodology, Section 4 shows the design specification of the project, Section 5 explains the implementation part, Section 6 involves the evaluation of this research and lastly Section 7 concludes this research with future work.

2 Related Work

Related work as regards to MCC offloading has been discussed in this section. This section is further split into various subsections for better understanding.

2.1 MCC Architecture

The human lifestyle has been drastically transformed by the mobile devices being as a crucial element in daily events. For their betterment, mobile features and applications are getting more complexed and also increasing. ML, AI, deep learning are some of the compute intensive tasks that are offered by the complex mobile applications to fulfil user needs but at the same time these applications require high energy and computation power to generate the output. This problem can be solved by offloading such tasks to powerful computation nodes.

Physical Layer and Application Layer are the two layers by which the architecture of MCC is made off. As mentioned [6], the most widely accepted architecture of MCC is shown in Figure 1. LTE, UMTS and GPRS are an expensive connections that mostly the mobile devices use to establish a connection to the internet [7]. They are costly because of its availability, location privacy etc. Establishing connection through Wi-Fi is another possibility, but both have their own pros and cons. As authors mentioned [8], physical layer helps the resource constrained

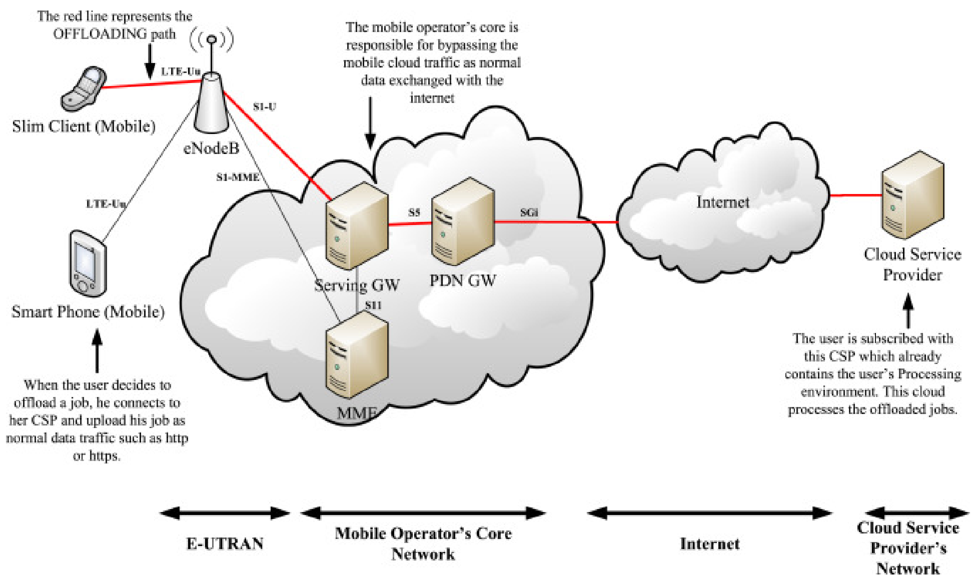

The red line represents the OFFLOADING path
LTE-Uu
Slim Client (Mobile)
eNodeB
S1-U
S1-MME
LTE-Uu
Smart Phone (Mobile)
When the user decides to offload a job, he connects to her CSP and upload his job as normal data traffic such as http or https.
The mobile operator's core is responsible for bypassing the mobile cloud traffic as normal data exchanged with the internet
Serving GW
S5
PDN GW
SGi
S11
MME
Internet
Cloud Service Provider
The user is subscribed with this CSP which already contains the user's Processing environment. This cloud processes the offloaded jobs.
E-UTRAN
Mobile Operator's Core Network
Internet
Cloud Service Provider's Network

Figure 1: Cloud Computing with Mobile Terminals Architecture [6].

devices by saving their computation power and energy. Application layer architecture comprises of decision-making framework and offloading technique. It generally concentrates on decision framework and code partitioning. This layer has to decide when and where to offload the task. In this research, application layer has been improved with the help of DME which handles the computationally intensive tasks based on device resources.

2.2 Detection of android resource usage

In order to decide whether to proceed with offloading or not, service must correctly evaluate computation power. To calculate the network and CPU consumption on the mobile devices, authors in this paper [9], have investigated the effectiveness and possibility of developing power-consumption-based sensors. Various power consumption metering methods have been evaluated by focusing on the data gathered from the mobile devices. It split into three categories. The number of tests has also been executed from security viewpoint.

As mentioned [10], low level measurements are provided by these two tools i.e., FEPMA (Fine-grained event-driven power meter for android) and Appscope. They both have been tied specifically to device model and considered as very invasive to kernel. PowerTutor is the tool mainly utilized for high-level measurements and to control the data of mobile device hardware into its models to display power consumption [11]. It contains six components: LED, GPS, CPU, Wi-Fi, audio, and cellular interfaces. Middle level measurement is the second last method of measurement as described by the authors [9], in which battery level and consumption is evaluated without establishing the connection to the kernel.

It's crucial to detect resource usage before applying any new algorithm or feature as it facilitates product to work properly. Most authors have promoted middle level measurement for task offloading and security.

2.3 Android Architecture

This research project concentrates on Android OS smartphone, but the logic and investigation can be utilized in any OS for future studies. Windows, iOS, blackberry, etc are some of the other OS but according to authors [12], Android platform is classified as more popular and user-friendly OS. This is an open-source Linux based OS. Authors in Samsung KNOX research

[13], have shown an effective evaluation of important spots in design and implementation of security in an android environment. Also, the security is really important for this research. Any developer can upload their application to Android Play Store that's why it is more prone to security vulnerabilities [12]. Authors have listed various options to protect the device through the utilization of MDM solution (mobile device management) [13]. Hardware implementation of ARM TrustZone is foundation of trust, and it can perform secure code offloading. Knox architecture has three modules: SEAndroid, TIMA (Trustzonebased Integrity Measurement Architecture), and Secure boot. On top of traditional Android OS and extensive VPN framework, this architecture offers additional encryption. This could be beneficial for this research, as it can assist with securing and deploying of data.

There are multiple approaches for application development as mentioned by authors [12] but Cross-platform is recognised as the more convenient one because applications can be installed on several platforms without any change in code. Cross-platform also has some sub-approaches. Model-Driven Approach (MDA) is the most widespread methodology which has been considered as in trend. In this methodology low-level problems need not to be dealt because is guided by the modelling events. It includes three fundamental models: PIM (Platform Independent Model), PSM (Platform Specific Model) and PIM to PSM model. A good understanding of the working architecture is very important.

2.4 Context aware code offloading

Over the last few years, due to extremely popular applications and services including IoT the deployment issue in mobile devices has become much more noticeable. The context of mobile device is valuable to consider while performing code offloading. The device context includes available network, battery, memory etc. It's necessary to have an understanding of offloading before implementing it. MobiCOP-IoT solution is a code offloading framework in IoT as described by the authors [14]. This framework is expected to also support offloading in android devices using cloud and/or fog system. Here Figure 2 illustrates an overview of MobiCOP-IoT workflow. The decision-making algorithm in this solution includes code and network profiler and is integrated on a client-side. Code profiler utilizes heuristic-based algorithm to calculate time taken by a given task to be executed. Network profiler checks network every 15 min to verify strength and bandwidth of a connection. After complete evaluation, a decision will be made either to execute task on cloud or on local machine or both simultaneously. Benchmarking has proven that offloading to cloud than mobile device offers better performance without any further cost.

Authors in these papers [15] and [16] have proposed dynamic approach-based context aware offloading frameworks. This document [15] proposes an approximation model which chooses cloud resources necessary for offloading intensive tasks. It also works with network quality and cloud computation power because current cloud resources offer uncertain performance. In [16], authors primarily focused on signal strength and network range to do effective offloading. Both frameworks finalize their decisions using dynamic partitioning during application runtime. Results [15] have proven a decreased execution time of about 6% to 96% and power utilization of about 60% to 90%. Rest [16] has also indicated appropriate results by offering better performance and less power consumption. Both presented really decent results but not robust enough to various offloading options.

As mentioned by the authors [17], deep learning in mobile devices is increasing its popularity. The problem with this approach [17] is as devices are resource constrained and difficulties could arise if tasks need to be offloaded and the device does not have enough computation power to allocate. As mentioned by the authors [18], future work of offloading tasks mainly focusses on fog computing. For IoT deployments and mobile devices, all standards are counted as legitimate factors. It also evaluates tech enablers for offloading tasks in wireless technology, which is essential for accurate task offloading. Second enablers are smart autonomous applications which

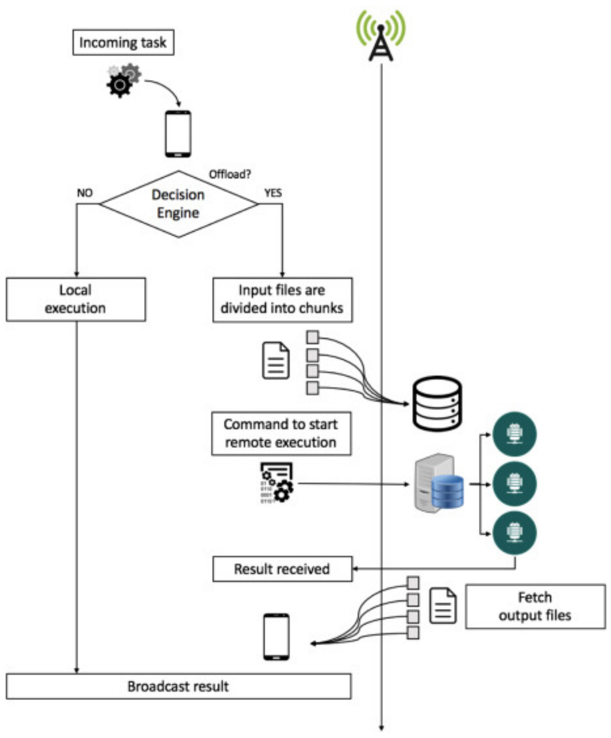


Figure 2: MobiCOP-IoT offloading diagram [14].

utilise ML and employ context aware services to allow concurrent offloading on various nodes. Lastly, this paper has also evaluated weaknesses and challenges that are important to offloading. As mentioned by authors, scalability and resource distribution are the two factors that can affect implementation. SLA is another stated issue. Other associated challenges could be fault tolerance, energy utilisation, service & monitoring etc.

Dpartner is a tool presented by authors [19] to support computation offloading. To find the bits worth for offloading, Dpartner tool analyses code of application, and then creates two artifacts that needed to be deployed on the device and on the cloud. This tool provides time improvement of 46 to 97 % and reduction in battery utilization of 27 to 83 %. The 1st realistic execution of offloading in Android OS is investigated by authors [20] known as Cuckoo. It is valuable to have a knowledge of present state of a device and framework.

2.5 Communication Protocol

The RESTful API is designed for the communication between components. It has the capability to cope with various forms of requests and capable to return a number of data formats. For microservice execution, authors [21] have compared the performance of RESTful API and RabbitMQ. Results have shown that RESTful works better for microservices execution with lowest latency.

In this research, AWS cloud application model is used for the cloud execution. AWS handles the autoscaling like when there is change in network traffic, it will automatically scale-in or scale-out to provide better performance to end user. The microservice is deployed at AWS Elastic Beanstalk. This makes it more secure, and it can be accessed through RESTful API requests.

2.6 Code Offloading Frameworks

This section gives a quick overview of several existing offloading frameworks. These frameworks usually come with several approaches and methodologies. Authors [22] have stated that current work on offloading follows three methods categorized as the full migration, granularity and partition algorithm, and pre-calculated offline partitions and making decision at run-time. AndroidOff is the offloading framework proposed in this paper [23]. This solution is focused on cost calculation. It is achieved through a forecast method which works by applying testcases to specific methods to validate time required by methods to run. Results have shown that execution time can be reduced up to 8% - 49% and decrease in power consumption could be up to 12% - 49%.. One downside of this methodology is that in most situations the testcases used might not produce results suitable for a real-world situation. A joint-scheduling approach has been proposed by authors [24] for offloading and also for organizing the order of offloading. This approach relies on network accessibility and uses parallel execution of tasks both on AWS EC2 instances and locally. The downside of this approach is it lowers the selection of apps where it can be applied to. Generally, significant results have been attained with decreased power consumption for applications with prolonged run-times.

Another offloading framework known as MobiCOP has been proposed by authors [25]. It concentrates on the problem i.e., replicability of existing offloading frameworks and also on reducing the execution time [26]. MobiCOP is made-up of public cloud execution environment, offloading decision-making engine and a communication handler. The results have indicated that this framework offers about 25 times and 17 times improved power saving and efficiency respectively. It also decreases execution time by using service application component. MobiCop's framework [25] has client library, it includes android service that connects platform components and supervises execution of new tasks. To transmit communications via XMPP through Google's firebase cloud messaging system, platform communication layer employs asynchronous data transmissions. Middleware, elastic cloud component, public interface, and android server environment are the four modules involved in server. The MobiCop architecture, in which QoS (module of Decision Engine) cares about transfer speed, profiling network-availability and latency. Code profiler keeps track of previous tasks execution and predicts run-time for upcoming tasks.

In conclusion of related work section, it is derived that, number of concerns are still around that need to be solved. The frameworks show decent outcomes but are still restricted in their approach and consequently can lead to minimal application. My proposed framework uses state-of-the-art method to decide whether to offload a task or to run it locally. This DME resides on a mobile device and run constantly to provide real-time response.

3 Methodology

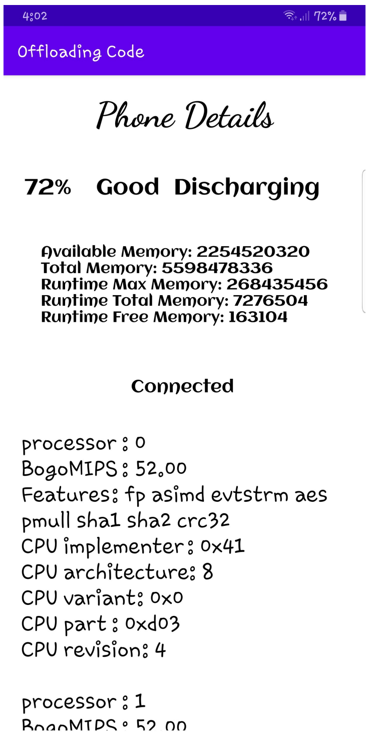
The research methods that have been applied while implementing this research are described in this section. Implementing offloading in this research involves android device and AWS cloud but before task execution, system calls a DME to check where to execute task either on a cloud or on a local device. The reason for proposing this algorithm is to work on the existing frameworks weaknesses. Technologies used in this research are Android studio IDE, Eclipse IDE, Postman, Java, XML, JSON, Spring Boot and AWS Elastic Beanstalk.

3.1 Dynamic Partitioning

In this research, dynamic partitioning is followed. In this type of partitioning, a decision is taken during application run-time. This dynamic approach has a lead over static approach as the mobile device context changes continuous. It has also been utilized in other previous related works [27]. *BroadcastReceiver* and *IntentFilter* are two Java methods that have used to achieve Dynamic partitioning. By using these methods, application makes a call to offloading DME to check where the task can be executed.

3.2 Getting real-time device context

It is important to get real-time stats of a mobile device to make offloading decision flawlessly as shown in Figure 3. It includes battery, memory, CPU and internet connection details. All these stats will then be passed to DME to decide where to execute the task. The DME changes its decision according to the change in device stats. All this happens on real-time basis.



```
Phone Details

72%   Good  Discharging

Available Memory: 2254520320
Total Memory: 5598478336
Runtime Max Memory: 268435456
Runtime Total Memory: 7276504
Runtime Free Memory: 163104

Connected

processor : 0
BogoMIPS : 52.00
Features: fp asimd evtstrm aes
pmull sha1 sha2 crc32
CPU implementer : 0x41
CPU architecture: 8
CPU variant: 0x0
CPU part : 0xd03
CPU revision: 4

processor : 1
BogoMIPS : 52.00
```

Figure 3: Device real-time statistics.

3.3 Decision-Making Engine (DME)

This engine is executed when button named DME is clicked in the application. The engine has Network Profiler, Memory Profiler, and Battery Profiler. The Network Profiler is used to check if the device is connected to the internet such as Wi-Fi or mobile data (3G/4G/5G). The Memory Profiler is employed to find current available RAM of the mobile device. This module is required to check if the device has adequate memory to execute task locally. Lastly, the Battery Profiler is used to examine the device battery level, battery health and also its current state. All these profilers i.e., Network Profiler, Memory Profiler, and Battery Profiler are the components that make the DME and are the foundation of the offloading decision process.

3.4 Execution of the task on a device.

The tasks can be executed locally on a device either as suggested by DME or as user preference. During local execution, user will provide an input and application will perform execution on the back. Once execution is finished, the application will provide a user an output. During local execution it is expected that application will consume more resources of a device like battery and memory.

3.5 Execution of the task on a cloud.

The tasks can also be executed on a cloud either as suggested by DME or as user preference. Before offloading a task to the cloud, application will counter check if device is still connected to a network or not. Spring Boot REST application is developed, which performs the same execution as execution of a task on local application. It is deployed on AWS Elastic Beanstalk. When user will provide an input, http POST request will be made. The application deployed on AWS Elastic Beanstalk will accept the input and perform the execution. Upon the completion of execution, result will be sent back to the app in JSON format. Finally, the required application will then parse this JSON response and display the results to the user.

3.6 Case Study

In this research, Fibonacci series has been implemented as a case study. The Fibonacci numbers are commonly denoted by Fn. The Fibonacci sequence is in which each number is the sum of the two preceding ones. The sequence commonly starts from 0 and 1. This series is implemented locally on a device for local execution and also on a cloud for cloud execution. Android Studio IDE is used for the development of mobile application including DME and Eclipse IDE for developing Fibonacci microservice for cloud execution. Java programming language is common between both but the microservice is developed with the help of Spring Boot REST framework. Performance metrics are being monitored and compared i.e., task completion time and battery, memory, CPU and network utilization. An android profiler is used to capture these metrics which summarizes the utilization of the resources by the case application.

4 Design Specification

The general architecture of the proposed solution is illustrated in Figure 4. It shows how the task offloading is being performed from local device to the cloud. The proposed solution uses AWS Elastic Beanstalk for cloud execution which integrates EC2 instance and S3 bucket. Elastic Beanstalk automatically handles resource provisioning, load balancing, auto scaling, and monitoring. Firstly, user should have a device which can run proposed application. Then, the DME will determine whether to execute task locally or offload it to the cloud based on various conditions. The DME uses the device current context to make decision. The tasks will be offloaded to the cloud through internet subject to it's availability i.e., Wi-Fi or cellular networks (3G/4G/5G).

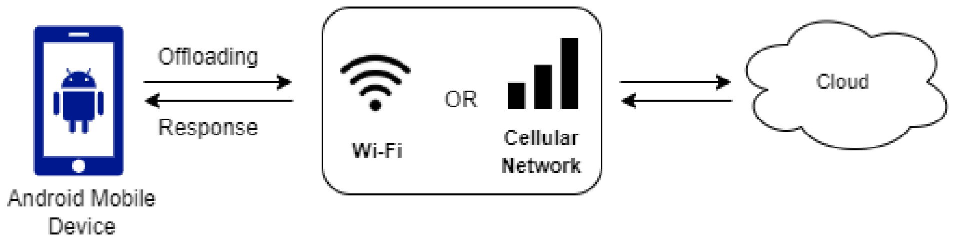


Figure 4: System Architecture of proposed solution.

4.1 System Components and Flow

The multiple modules and flow of this proposed solution is shown in Figure 5. Remote task execution and mobile device are the two main components in this approach. Offloading requests are made by the application on the mobile device. Once the request has been made, the DME will use *BroadcastReceiver* and *IntentFilter* methods during runtime on real-time bases. In this proposed solution, the device current context is composed of Network, Memory and Battery Profiler. They verify if the device has enough resources to execute task locally or not.

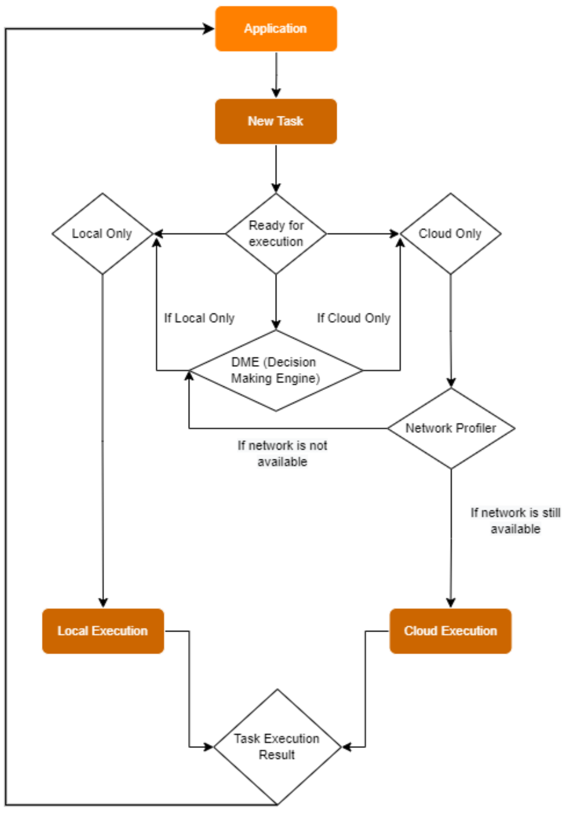


Figure 5: Overview of System Components and Flow.

4.2 Context Aware Offloading

The proposed algorithm is illustrated in Figure 6. Based on the device current context, the DME gives recommendation of where to execute task. There is not any default offloading choice. The application allows user to select one of the three options i.e., Local Only, Cloud Only and DME. There is another option named Information, it enables user to see real-time context of their device. While implementing this algorithm number of methods have been used to get device current context. If mobile device doesn't have enough resources and task is ready to offload, the final check will then be performed by Network Profiler to verify if the network is still available. Subject to network availability, the OFFLOAD_RECOMMENDATION can be switched from CLOUD to LOCAL execution.

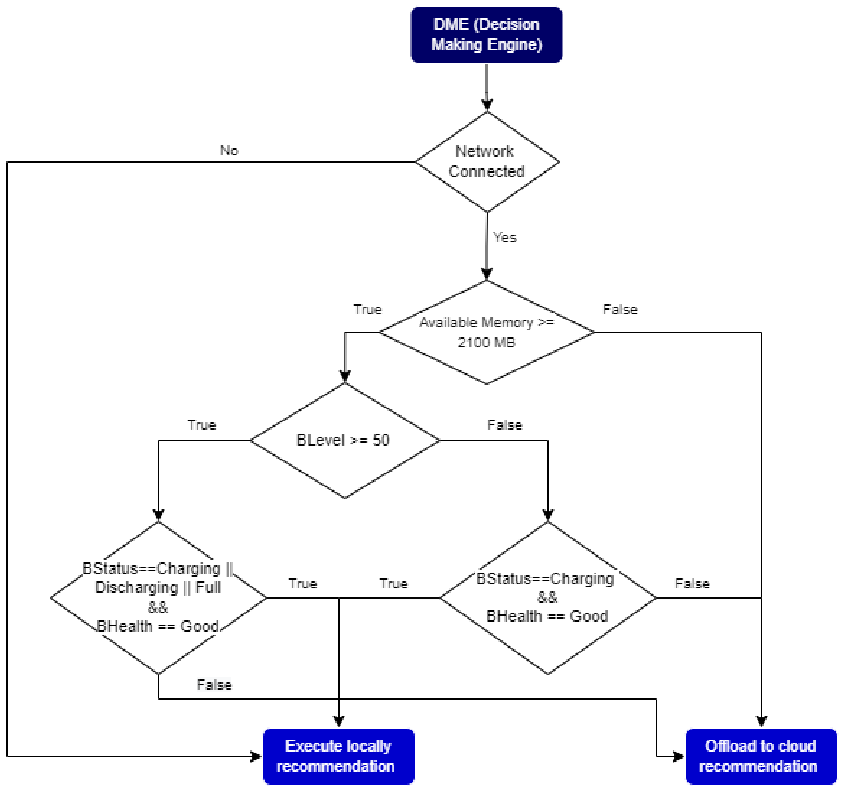
4.3 Decision-Making Engine (DME)

The DME contains the resource provisioning algorithm as shown in Figure 7. The DME executes the algorithm based on the output from the context monitor. The DME is framed in such a way that when there is no network, task execution is performed locally and if network connection is available, number of other checks will be performed before initialising the decision whether the cloud offloading can be performed or not. Here, other checks include verifying other parameters such as available memory, current battery level, battery status and battery health.

Algorithm 1 DME

```
procedure MAKEOFFLOADRECOMMENDATION(context)


    $OFFLOAD\_RECOMMENDATION \leftarrow Null$
    $networkProfiler \leftarrow new\ NetworkProfiler(context)$
    $batteryProfiler \leftarrow new\ BatteryProfiler(context)$
    $memoryProfiler \leftarrow new\ MemoryProfiler(context)$


    if $networkProfiler.getNetworkIsConnected() == false$ then
        return $OFFLOAD\_RECOMMENDATION \leftarrow LOCAL\_EXECUTE\_RECOMMENDATION$

    if $networkProfiler.getNetworkIsConnected() == true$ then

        if $memoryProfiler.getAvailableRAM() >= 2100$ then

            if $batteryProfiler.getStatus() == Charging\ AND\ batteryProfiler.getLevel() >= 50\ AND\ batteryProfiler.getHealth() >= Good$ then
                return $OFFLOAD\_RECOMMENDATION \leftarrow LOCAL\_EXECUTE\_RECOMMENDATION$

            if $batteryProfiler.getStatus() == Discharging\ AND\ batteryProfiler.getLevel() >= 50\ AND\ batteryProfiler.getHealth() >= Good$ then
                return $OFFLOAD\_RECOMMENDATION \leftarrow LOCAL\_EXECUTE\_RECOMMENDATION$

            if $batteryProfiler.getStatus() == Full\ AND\ batteryProfiler.getLevel() >= 50\ AND\ batteryProfiler.getHealth() >= Good$ then
                return $OFFLOAD\_RECOMMENDATION \leftarrow LOCAL\_EXECUTE\_RECOMMENDATION$

            if $batteryProfiler.getStatus() == Charging\ AND\ batteryProfiler.getLevel() < 50\ AND\ batteryProfiler.getHealth() >= Good$ then
                return $OFFLOAD\_RECOMMENDATION \leftarrow LOCAL\_EXECUTE\_RECOMMENDATION$

            else
                return $OFFLOAD\_RECOMMENDATION \leftarrow CLOUD\_EXECUTE\_RECOMMENDATION$
        else
            return $OFFLOAD\_RECOMMENDATION \leftarrow CLOUD\_EXECUTE\_RECOMMENDATION$



    return $OFFLOAD\_RECOMMENDATION$
```

Figure 6: DME Algorithm.



DME (Decision Making Engine)
Network Connected
No
Yes
Available Memory >= 2100 MB
True
False
BLevel >= 50
True
False
BStatus==Charging || Discharging || Full && BHealth == Good
True
False
BStatus==Charging && BHealth == Good
True
False
Execute locally recommendation
Offload to cloud recommendation

Figure 7: DME.

5 Implementation

5.1 DME (Algorithm)

The resource provisioning algorithm i.e., DME is embedded in the application to consider the current network, memory and battery status of the device before finalising the offloading decision. The algorithm is activated once the button named DME present in the home screen of application is clicked. As shown in Algorithm 1, several IF statements will be executed and if offloading cannot take place because of network, memory or battery of the device, the local execution of the Fibonacci series will take place.

5.2 Communication Module

This module is accountable for establishing the connection between the server-side and the client-side for communication. For cloud offloading, the mobile device should be connected to the network to perform HTTP GET request. The Fibonacci cloud microservice is developed using Java Spring boot REST framework and deployed in AWS Elastic Beanstalk. The application uses Volley API for sending and receiving data in JSON format.

5.3 Technologies used

The number of technologies used for the completion of this project are Android studio IDE, Eclipse IDE, Postman, Java, XML, JSON, Spring Boot, Git and AWS Elastic Beanstalk. For the development of mobile application including DME and Fibonacci local execution, Android studio IDE has been used in which Java programming language used for developing backend features and XML for designing frontend layouts. Git has also been integrated for the version control in case of any errors. For the cloud execution, the Fibonacci series microservice has been developed in Eclipse IDE using Java Spring boot REST framework. Postman has been used to test GET requests. The microservice has been deployed in a cloud called AWS Elastic Beanstalk and responses of GET request are always in JSON format. Volley API has been used for sending GET requests and parsing responses to display result back to user in an application.

6 Evaluation

The number of experiments has been performed to evaluate this research work. The findings and how experiments support the objectives of this research work have also been discussed. The execution of Fibonacci series takes place on a cloud server and on a local device. The conclusion is drawn based on the utilization of mobile device resources and execution time on the two environments with different workloads (inputs). Since the time to execute each workload varies frequently therefore three readings are taken, and their average is considered as the final result. Android studio profiler has been used in the evaluation process. Number of android devices with various parameters (battery, network, and memory) have also been used in order to test context aware DME.

6.1 Experiment 1 / Fibonacci series - Local execution

Local execution has been performed on a device called Galaxy Note 8. While performing this experiment it had 70% of battery level with good health and 2500 MB of free memory. Each input is provided three times and then average time is calculated. Figure 8 shows the Local execution times of each input i.e., 45,47 and 49. 49 has the highest execution time and consumed more resources of device for a longer period of time.

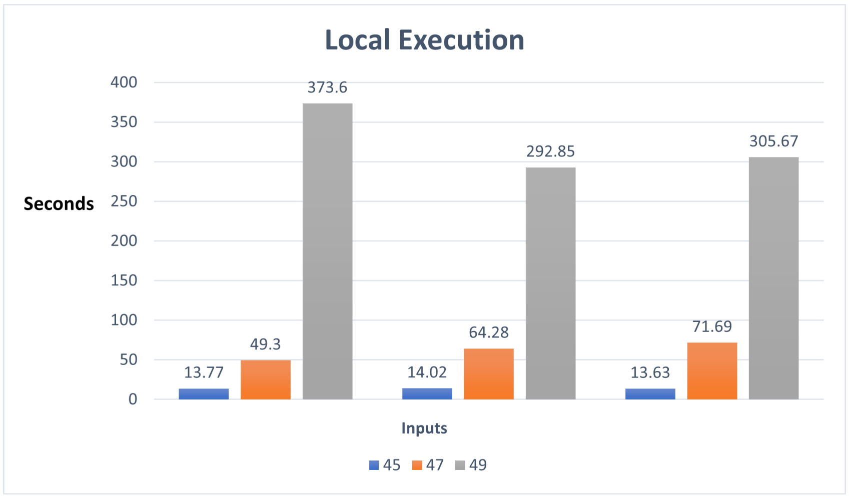


Figure 8: Local execution time of each input.

The average local execution time of each value is shown in Figure 9. The result depicts that there is dramatic increase in execution time between 47 and 49 than 45 and 47.

Android studio profiler has been used for the analysis of resource utilization of a device while performing local execution. As illustrated in Figure 10, the local execution of Fibonacci series occupied 12% of CPU and 107.8 MB of Memory. Network usage was 0 B/s for both sending and receiving because execution happened locally. The energy utilization varied between light and medium. In local execution, full load is on a device CPU.

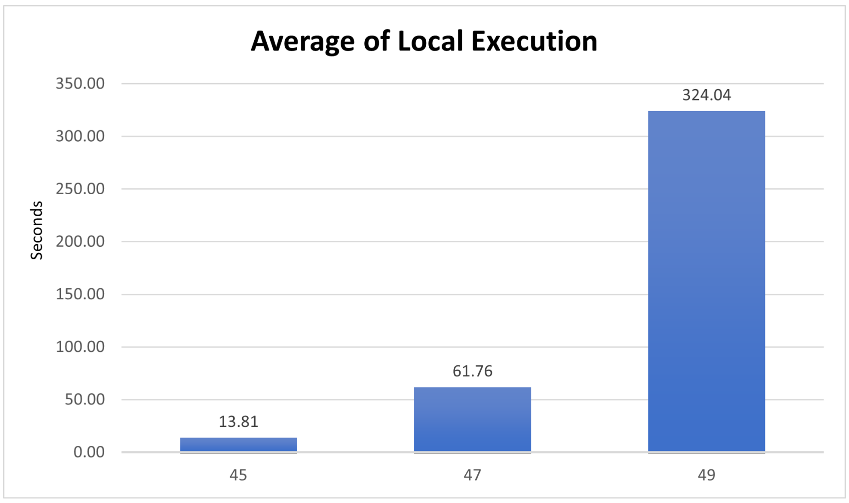


Figure 9: Average time of each input.

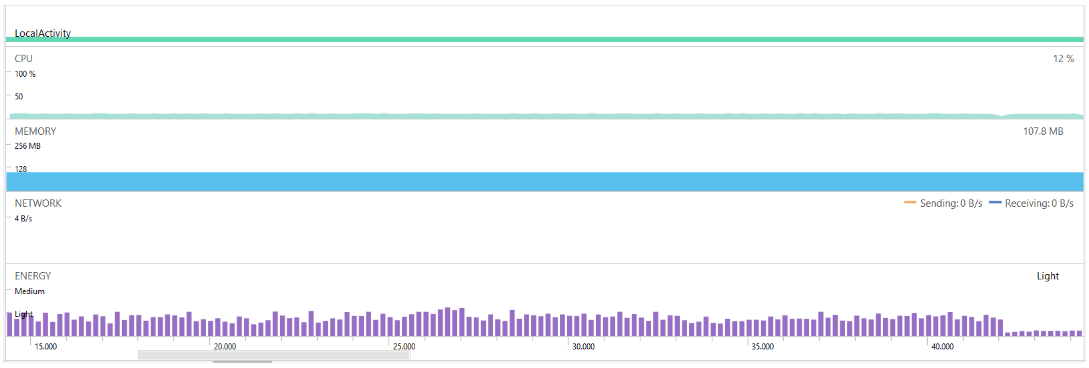


Figure 10: Android profiler values of local execution.

6.2 Experiment 2 / Fibonacci series - Cloud execution

Cloud execution has been performed with the help of AWS Elastic Beanstalk. It automatically handles resource provisioning, load balancing, auto scaling, and monitoring. In cloud execution process an android device is only responsible for accepting user input, making an API call, and displaying the final result. The whole execution process is done up on a cloud. Even for this experiment, each input is provided three times and then average time is calculated. Figure 11 shows the Cloud execution times of each input i.e., 45,47 and 49. 49 has the highest execution time than rest two.

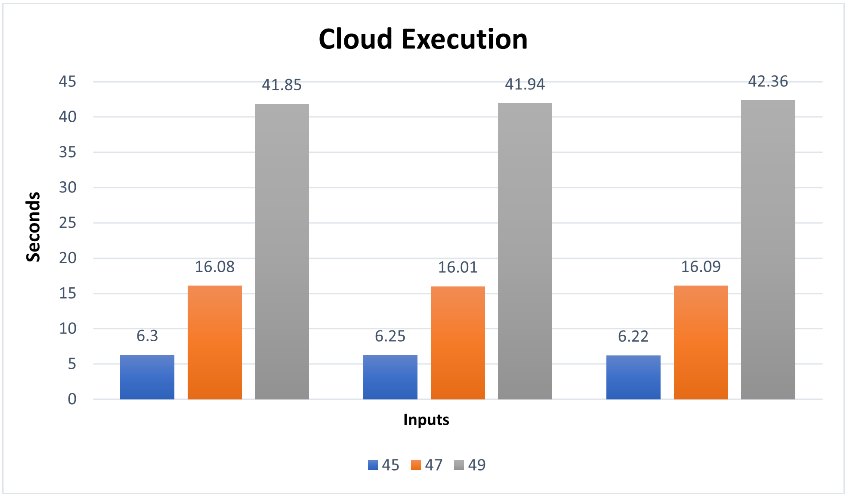


Figure 11: Cloud execution time of each input.

The average cloud execution time of each value is shown in Figure 12. The result depicts that there is more execution time difference between 47 and 49 than 45 and 47.

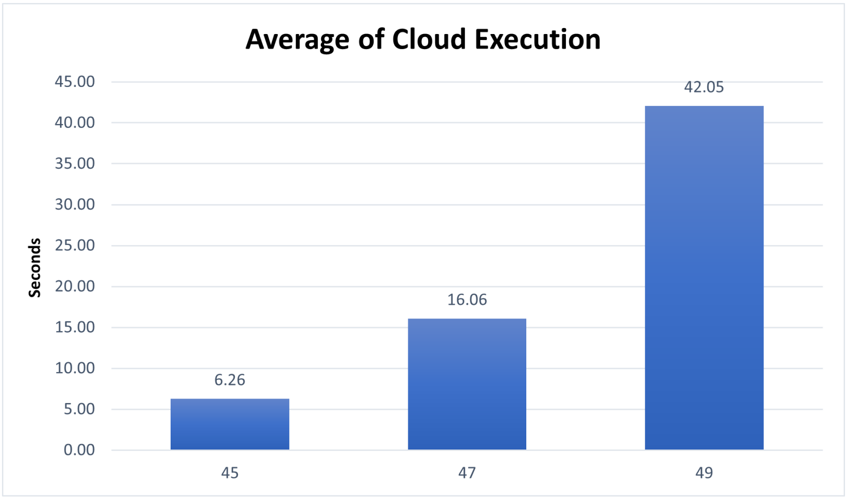


Figure 12: Average time of each input.

Android studio profiler has also been used for the analysis of resource utilization of a device while performing cloud execution. As illustrated in Figure 13, cloud execution of Fibonacci series occupied 0% of CPU and 62.2 MB of Memory. Network usage was 2.9 B/s for sending and 0.9 B/s for receiving because execution happened on a cloud. The energy utilization varied between none and light. In cloud execution, there is no load on a device CPU.

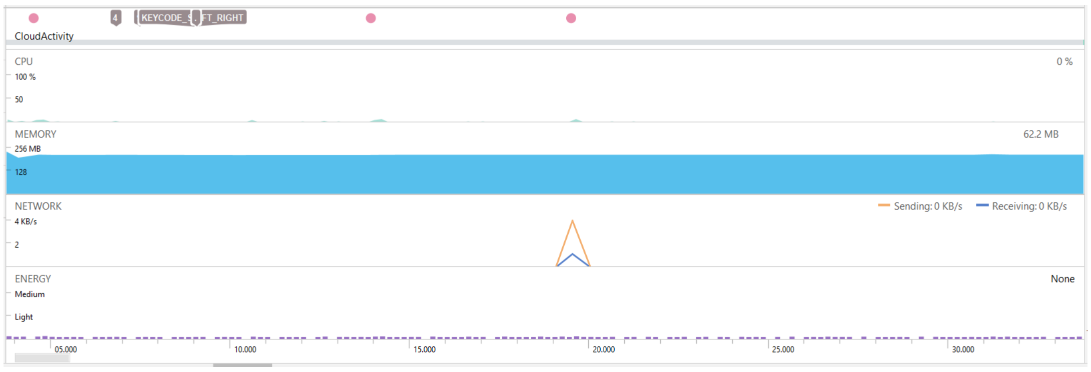


Figure 13: Android profiler values of cloud execution.

The results indicate that compared to local execution, offloading to cloud server drastically reduces the execution time and it also leads to performance improvement in mobile device. Offloading tasks to cloud free-up space in mobile device for tasks that cannot be offloaded.

6.3 Experiment 3 / DME

The evaluation of DME is performed in this section. Number of android devices with different context parameters have been used in order to determine if proposed context aware DME is functioning properly or not. Table in Figure 14 shows the DME execution results.

From Figure 14, it can be seen that, proposed DME is working correctly and producing accurate decisions centred on the changing device context.

6.4 Discussion

The experiment is performed on both components of the application i.e., local and cloud, by giving three different input values. It can be seen based on the results achieved, that the execution time of each workload is significantly lower when offloaded to the cloud server than local execution. Additionally, if the higher hardware specification of EC2 instance will be used, the time to execute each workload in the cloud could be even more lowered. In this research project free tier EC2 instance has been used. The execution time is evaluated based on the computation time taken by the workload to be executed in both local and cloud. The results also take some time to get displayed on the mobile device, but this research only focuses on the execution time and not the display time. CPU, memory, battery, and network utilization have also been assessed, and it's been proved that cloud execution is faster and take very less mobile device computation power. Another experiment has been done to test the functionality of DME and the results in table 1 have proven that it is working as expected. The proposed DME makes the right offloading decision based on the device current context. While performing this experiment, number of different devices with various context have been used. The DME changes its decision on real-time bases as the context of mobile device changes.

The results attained have indicated to be in line with results found from previous related work and comparison between this proposed approach and other related work has proven that this approach is better in the sense that it makes decision on real-time bases and considers memory, battery, and network fully before finalizing the decision. This research paper has potential to contribute in the field of cloud computing because it is revealing some of the cloud services that can be employed in mobile application development.

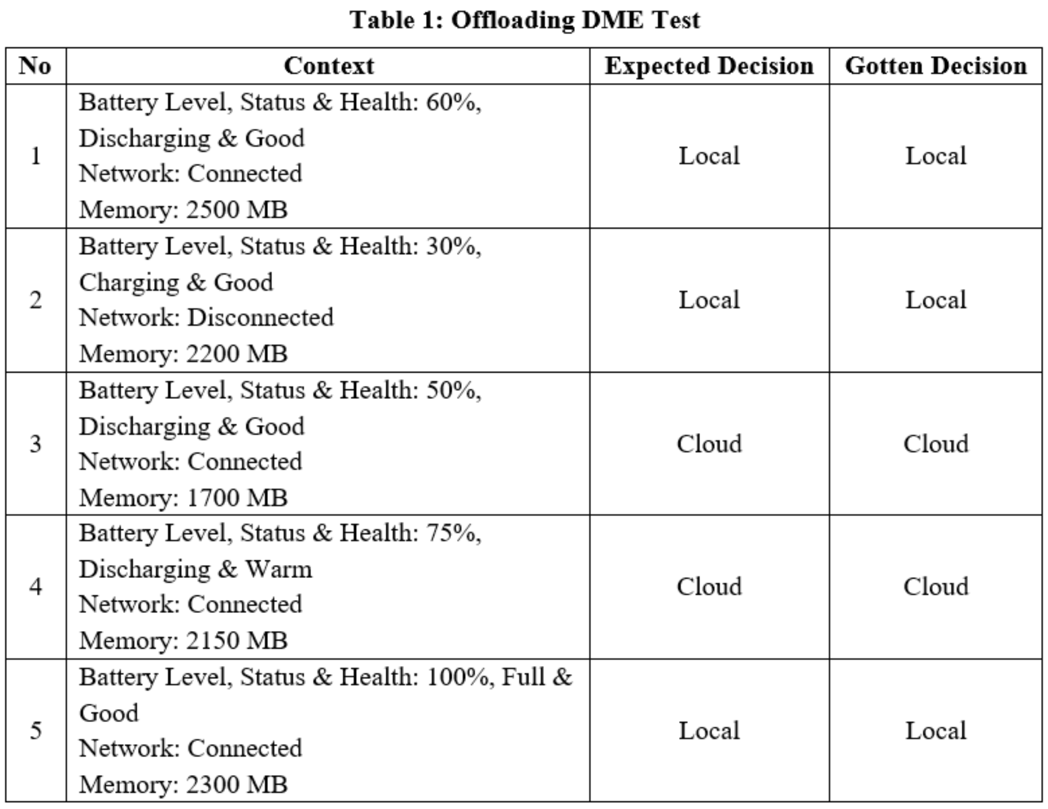


Table 1: Offloading DME Test

No	Context	Expected Decision	Gotten Decision
1	Battery Level, Status & Health: 60%, Discharging & Good Network: Connected Memory: 2500 MB	Local	Local
2	Battery Level, Status & Health: 30%, Charging & Good Network: Disconnected Memory: 2200 MB	Local	Local
3	Battery Level, Status & Health: 50%, Discharging & Good Network: Connected Memory: 1700 MB	Cloud	Cloud
4	Battery Level, Status & Health: 75%, Discharging & Warm Network: Connected Memory: 2150 MB	Cloud	Cloud
5	Battery Level, Status & Health: 100%, Full & Good Network: Connected Memory: 2300 MB	Local	Local

Figure 14: Offloading DME Test

7 Conclusion and Future Work

In this research, the framework for offloading the compute intensive tasks of an android application is developed named DME (Decision Making Engine). In this proposed framework, the task execution location is decided based on the current context of the device. Tasks can either be offloaded to the cloud or execute it locally. AWS Elastic Beanstalk including EC2 and S3 is used as the cloud server. An android application and a microservice with the Fibonacci series is built to evaluate this experiment. The complex components while developing this project include, making offloading decision based on the device current context and cloud execution. The execution time and, CPU, memory, network, and battery utilization of a device are considered as the parameters to evaluate. The results attained show that offloading task to the cloud significantly reduces the execution time as compared to the local execution. This also helps to reduce CPU and memory utilization, and to increase the battery life of the device. This research project can be a comprehensive progress in task offloading.

The research work of the proposed solution will be further extended to the analysis of cloud server context and comparison between available computation power of local device and cloud before concluding offloading decision. More features of network profiler can also be employed in future work because network plays a vital role in offloading a task. Network parameters such as signal strength and bandwidth need to be considered for making offloading decision.

References

[1] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, and Varghese, "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Computing Surveys*, vol. 51, no. 5, Nov. 2018, jCR Impact Factor 2021: 10.282. [Online]. Available: https://doi.org/10.1145/3241737

[2] T. H. Noor, S. Zeadally, A. Alfazi, and Q. Z. Sheng, "Mobile cloud computing: Challenges and future research directions," *Journal of Network and Computer Applications*, vol. 115, pp. 70–85, Aug. 2018, jCR Impact Factor 2021: 6.281. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804518301504

[3] K. Akherfi, M. Gerndt, and H. Harroud, "Mobile cloud computing for computation offloading: Issues and challenges," *Applied Computing and Informatics*, vol. 14, no. 1, pp. 1–16, Jan. 2018, jCR Impact Factor 2021: 6.82. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210832716300400

[4] B. Zhou and R. Buyya, "Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions," *ACM Comput. Surv.*, vol. 51, no. 1, jan 2018, jCR Impact Factor 2021: 10.282. [Online]. Available: https://doi.org/10.1145/3152397

[5] F. Gu, J. Niu, and Z. Qi, "Partitioning and offloading in smart mobile devices for mobile cloud computing," *Network and Computer Applications*, vol. 119, pp. 83–96, oct 2018, jCR Impact Factor 2021: 6.281. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804518302170

[6] J. Bou Abdo and J. Demerjian, "Evaluation of mobile cloud architectures," *Pervasive and Mobile Computing*, vol. 39, pp. 284–303, aug 2017, jCR Impact Factor 2021: 3.453. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574119216304114

[7] B.-G. Chun and S. Ihm, "Clonecloud: Elastic execution between mobile device and cloud," in *the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: Association for Computing Machinery, Apr. 2011, p. 301–314, CORE2021 Rank: A. [Online]. Available: https://doi.org/10.1145/1966445.1966473

[8] R. Buyya and A. Qureshi, "Application partitioning algorithms in mobile cloud computing," *Network and Computer Applications*, vol. 48, pp. 99–117, feb 2015, jCR Impact Factor 2021: 6.281. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804514002161

[9] A. Merlo, M. Migliardi, and P. Fontanelli, "On energy-based profiling of malware in android," in *2014 International Conference on High Performance Computing Simulation (HPCS)*, Sep. 2014, pp. 535–542, CORE2021 Rank: B.

[10] K. Kim, D. Shin, Q. Xie, Y. Wang, M. Pedram, and N. Chang, "Fepma: Fine-grained event-driven power meter for android smartphones based on device driver layer event monitoring," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, Apr. 2014, pp. 1–6, CORE2021 Rank: B.

[11] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Apr. 2010, pp. 105–114, CORE2021 Rank: C.

[12] A. Sarkar, A. Goyal, D. Hicks, D. Sarkar, and S. Hazra, "Android application development: A brief overview of android platforms and evolution of security systems," in *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Dec. 2019, pp. 73–79, CORE2021 Rank: N/A.

[13] U. Kanonov and A. Wool, "Secure containers in android: The samsung knox case study," in *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, p. 3–12, CORE2021 Rank: A*. [Online]. Available: https://doi.org/10.1145/2994459.2994470

[14] J. I. Benedetto, L. A. González, P. Sanabria, A. Neyem, and J. Navón, "Towards a practical framework for code offloading in the internet of things," *Future Generation Computer Systems*, vol. 92, pp. 424–437, mar 2019, jCR Impact Factor 2021: 7.187. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18302310

[15] X. Chen, S. Chen, X. Zeng, X. Zheng, Y. Zhang, and C. Rong, "Framework for context-aware computation offloading in mobile cloud computing," *Journal of Cloud Computing*, vol. 6, jan 2017, jCR Impact Factor 2021: 5.711. [Online]. Available: https://doi.org/10.1186/s13677-016-0071-y

[16] A. Islam, A. Kumar, K. Mohiuddin, S. Yasmin, M. A. Khaleel, and M. R. Hussain, "Efficient resourceful mobile cloud architecture (mrarsa) for resource demanding applications," *Journal of Cloud Computing*, vol. 9, feb 2020, jCR Impact Factor 2021: 5.711. [Online]. Available: https://doi.org/10.1186/s13677-020-0155-6

[17] M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, and X. Liu, "Deepwear: Adaptive local offloading for on-wearable deep learning," *IEEE Transactions on Mobile Computing*, vol. 19, no. 2, pp. 314–330, feb 2020, jCR Impact Factor 2021: 5.577. [Online]. Available: https://ieeexplore.ieee.org/document/8618364

[18] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for iot: Review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, vol. 87, pp. 278–289, oct 2018, jCR Impact Factor 2021: 7.187. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18301973

[19] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring android java code for on-demand computation offloading," in *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA '12. New York, NY, USA: Association for Computing Machinery, Oct. 2012, p. 233–248, CORE2021 Rank: A. [Online]. Available: https://doi.org/10.1145/2384616.2384634

[20] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*, M. Gris and G. Yang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 59–79, CORE2021 Rank: N/A. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-29336-8_4

[21] X. J. Hong, H. Sik Yang, and Y. H. Kim, "Performance analysis of restful api and rabbitmq for microservice web application," in *International Conference on Information and Communication Technology Convergence (ICTC)*, Jun. 2018, pp. 257–259.

[22] Y. Zhang, H. Liu, L. Jiao, and X. Fu, “To offload or not to offload: An efficient code partition algorithm for mobile cloud computing,” in *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, Mar. 2013, pp. 80–86, CORE2021 Rank: B. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6483660

[23] X. Chen, J. Chen, B. Liu, Y. Ma, Y. Zhang, and H. Zhong, “Androidoff:offloading android application based on cost estimation,” *Journal of Systems and Software*, vol. 158, p. 110418, dec 2019, jCR Impact Factor 2021: 2.829. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412121930192X

[24] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, “Optimal joint scheduling and cloud offloading for mobile applications,” *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 301–313, apr 2019, jCR Impact Factor 2021: 5.938. [Online]. Available: https://ieeexplore.ieee.org/document/7463066

[25] J. I. Benedetto, A. Neyem, J. Navon, and G. Valenzuela, “Rethinking the mobile code offloading paradigm: From concept to practice,” in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, Jul. 2017, pp. 63–67, CORE2021 Rank: A. [Online]. Available: https://ieeexplore.ieee.org/document/7972719

[26] K. E. Psannis, G. Benedetto, José I.and Valenzuela, P. Sanabria, A. Neyem, J. Navón, and C. Poellabauer, “Mobicop: A scalable and reliable mobile code offloading solution,” *Wireless Communications and Mobile Computing*, vol. 2018, jan 2018, jCR Impact Factor 2021: 2.336. [Online]. Available: https://doi.org/10.1155/2018/8715294

[27] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, “mcloud: A context-aware offloading framework for heterogeneous mobile cloud,” *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 797–810, oct 2017, jCR Impact Factor 2021: 8.216. [Online]. Available: https://ieeexplore.ieee.org/document/7362223