Mobile Offloading technique for Latency-sensitive and Computational-intensive task

MSc Research Project
Cloud Computing

Niranjan Karunanithi
Student ID: X18177727

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Niranjan Karunanithi
Student ID:	X18177727
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Manuel Tova-Izquierdo
Submission Due Date:	17/08/2020
Project Title:	Mobile Offloading technique for Latency-sensitive and Computational-intensive task
Word Count:	6211
Page Count:	19

Signature:	
Date:	28th September 2020

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Mobile Offloading technique for Latency-sensitive and Computational-intensive task

Niranjan Karunanithi
X18177727

Abstract

The idea of Mobile cloud computing is to utilise the resources of the powerful computing nodes for the improvement of performance in resource constraint devices. The offloading decision has been taken in various aspects of the device. In this paper, the decision making engine will offload both computational intensive task and latency sensitive task based on the device current parameters by considering different decision factor for respective type of task. Offloading computational intensive task will free up memory space in the device which improves the device performance but offloading latency sensitive task improves device performance with increased latency. Decision making engine decide when to offload and where to offload task by considering parameters such as user preference, type of network connectivity, type of charging, complexity of task. Decision engine works with different algorithm for the two types of tasks which increases the overall performance the device.

1 Introduction

In the modern world, number of mobile device increasing predominantly and also the need for smart devices. The major drawback or threat to this devices are the finite amount of resources allocated to them, in other words unexpansive capacity of device which is explained by Buyya et al. (2019). To overcome this issue, Mobile cloud computing has been introduced in resource constrained devices. The main key idea of Mobile cloud computing is to offload identified tasks from mobile to cloud. Execution of this identified tasks perform better in cloud than on local device which make space for some other critical task that can only be executed in the local device. The most crucial part in this process flow is decision making for offloading which is discussed by Noor et al. (2018), it has to make accurate decision for offloading. The decision making framework lies within mobile application which will decide which task can be offloaded and when it can be offloaded. Many decision making framework has been introduced based on various parameters which can able to handle only specific types of task such as computational intensive task or latency sensitive task.

In this paper, we have formulated decision making engine in Android application which will handle both delay sensitive task and computational intensive task. Proposed framework will consider different decision factor for this two types of task. Tasks in android application are replicated as microservices in AWS lambda functions and can accessed through REST API calls through AWS API Gateway. AWS API gateway is integrated with AWS lambda functions, so that it can be accessed through API calls.

If the device parameters favours offloading, the decision making framework will offload task to cloud. Offloaded task will be executed in the AWS lambda functions when the decision engine request for offloading. The results shows offloading task to cloud improve the performance of the device.

2 Related Work

2.1 Mobile Cloud Architecture

Mobile device has become a vital part in day-to-day activities which drastically changes the human lifestyle. Mobile applications also increasing in count for the betterment of humans daily activities. Complex mobile application enforced with deep learning and machine learning will provide accurate results and perform better to fulfil the user requirement. But the complex application require more computing power and energy to interpret the results. So Mobile cloud computing has been introduced to offload the complex task to powerful computing node.

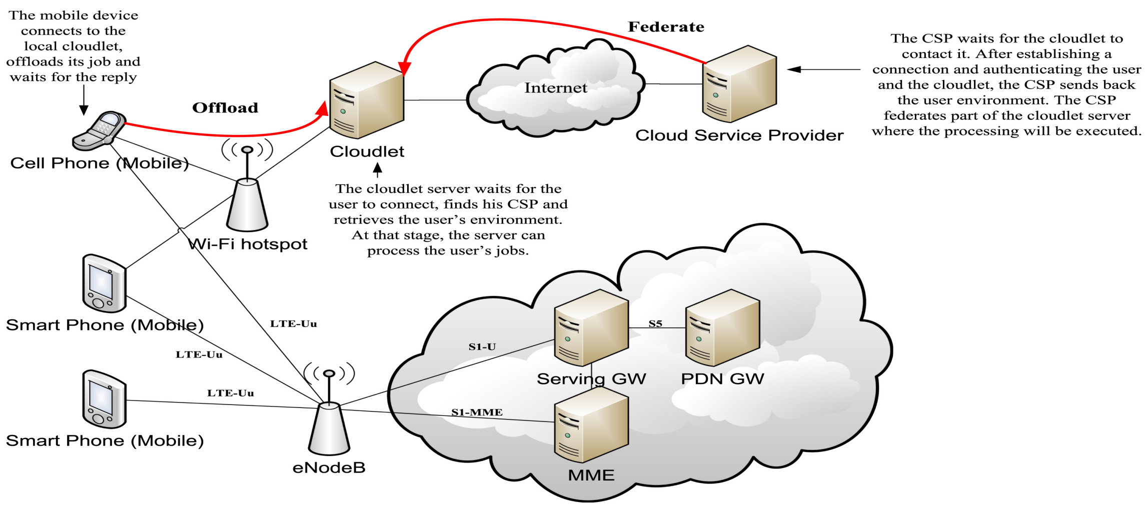

The mobile device connects to the local cloudlet, offloads its job and waits for the reply
Offload
Federate
Internet
The CSP waits for the cloudlet to contact it. After establishing a connection and authenticating the user and the cloudlet, the CSP sends back the user environment. The CSP federates part of the cloudlet server where the processing will be executed.
Cloud Service Provider
Cell Phone (Mobile)
Cloudlet
The cloudlet server waits for the user to connect, finds his CSP and retrieves the user's environment. At that stage, the server can process the user's jobs.
Wi-Fi hotspot
Smart Phone (Mobile)
LTE-Uu
LTE-Uu
S1-U
S5
Serving GW
PDN GW
LTE-Uu
S1-MME
Smart Phone (Mobile)
eNodeB
MME

Figure 1: Cloudlet Architecture by Abdo and Demerjian (2017)

Mobile cloud Architecture has two layers namely Physical Layer and Application Layer. There is no standard architecture for Mobile cloud computing so far, stated by Abdo and Demerjian (2017). But, widely accepted Physical layer architecture is Cloudlet Architecture. The key idea of Cloudlet is to provide computing and storage capabilities to nearby mobile devices by offloading mobile tasks. The powerful computing node is connected to mobile devices with high speed communication channel to avoid latency issues. Mobile devices will offload complex task to nearby computing node for computation and get back the results. It helps the resource constrained device to save energy and processing memory is discussed by Liu et al. (2015). In this paper, cloudlet physical layer architecture is considered as it perform better than others.

Offloading techniques and decision making framework will fall under Application layer architecture. Application layer mainly focuses on code partitioning and decision framework. This application layer has to decide where to offload the task and when to offload the task. In this paper, application layer has been enhanced with novel decision framework which handle both latency sensitive and computational intensive tasks and code will get offloaded to AWS lambda functions for cloud execution.

2.2 Scenarios for offloading

Depends on various factors code will be offloaded from resource constrained devices to rich computing node. The decision will be made based on the current device profile and network connectivity. Offloading tasks to cloud is mainly due to deficiency in resources of mobile devices and more energy consumption is analysed by Masip-Bruin et al. (2016). Offloading computational intensive task to cloud will consume less energy in resource constrained device is evaluated by Huang et al. (2017) which will increase the computation capabilities of mobile devices with minimum latency. For delay sensitive tasks, the tasks will be offloaded to nearest node to keep minimum latency and it will fall under fog computing.

Osanaiye et al. (2017) offload task from one computing node to another if it reaches maximum computing capabilities and it can be achieved only in distributed computing environment. It proves that this architecture is fault tolerant in which if one computing node fails or refuses to accept request, same request will be executed in the nearby computing node.

Another major drawback of resource constrained device is limited storage capabilities. Elgazar et al. (2018) proposed a model to store frequently accessed data in local device and rarely used data in cloud environment for future purposes which will free up memory in resource constrained device. Some of the task will handle sensitive data such as health data which cannot be send to public cloud without user concern due to security issues. So user permission is required to offload task to cloud and it is explained by Aazam et al. (2016)

2.3 Offloading framework

Offloading framework can be described as how the code offloaded to cloud. Many Offloading framework has been introduced with different decision factors. MAUI framework which is invented by Cuervo et al. (2010) forms base framework for other evolved frameworks. In the model, the code partition is done up to class level instead of cloning whole application in cloud. The decision making algorithm resides inside the application will decide whether the task can be executed locally or in cloud. All tasks in the mobile application cannot be offloaded to cloud, some of them are depend on I/O device or sensors of local device. It's programmers responsibilities to identify tasks that can be independently executed in cloud. The author proved that offloading complex task in cloud reduce energy consumption in local device.

CloneCloud by Chun et al. (2011) is the another offloading framework which reduces the burden of programmers to change the source code for offloading compared to MAUI framework. The author achieves thread level granularity in code partitioning. The static code partitioning is done in offline and cloned in cloud server. The author claims the proposed model is multithreading but the task which executes locally has to wait for remotely executed threads to complete so it fails to comply with author claim and it affects device performance.

ThinkAir by Kosta et al. (2012) is another popular framework in Mobile cloud computing. The idea of ThinkAir framework is to annotate the offloadable task in the development phase of application and offloading decision will be made during the execution of task. The author achieves method-level granularity in code partitioning which is better than CloneCloud as it did not create any wait time for cloud execution. The decision for offloading is taken by execution controller which resides inside the application based on device current status and network connectivity and author showed better results in the device performance.

Another framework is introduced by Kemp et al. (2010) for Android application and author proved that execution in the cloud is energy efficient with better device overall performance. Major drawback of this framework is platform dependent and it is restricted only to Android application. Montella et al. (2017) offloading is quite straightforward in which programmers should annotate offloadable task in the development phase itself. During execution, if the annotated task called for offloading, control send to RAPID compiler which resides inside the application. RAPID compiler will provide required code for offloading and it comes under CPU computational offloading. For GPGPU computational offloading, as the code were in native CUDA functions developer has to call RAPID's API to provide wrapper class for the functions further it will get offloaded to cloud.

Benedetto et al. (2019) focused on the execution time alone rather than energy consumption. The author used records for each task completion time in both local and cloud environment. With the heuristics based algorithm, decision making engine approximates the task execution time in local and cloud. Decision will be made to offload task if network bandwidth is available with less execution time in cloud. With three different types of mode in the application namely Optimistic mode, Concurrent mode and Cloud only mode, user can prefer mode in their choice. Optimistic mode will execute all process in local device and decision making framework wont try to offload any task. In concurrent mode, decision making framework will check for time approximation and decide based on less time for task execution. Cloud only mode will prefer always cloud if network bandwidth is available.

2.4 Offloading techniques

Zhao et al. (2016) mainly focused on the network latency and energy consumption.The author compared latency of fog node and cloud. The proposed model will calculate energy consumption in cloud, fog and local device. Decision making algorithm will make decision based on the energy consumption. On contrast, Fricker et al. (2016) discussed about the task will get offloaded between nodes in distributed environment. The proposed model will achieve load balancing by offloading tasks from crowded node to near by available node and it become fault tolerant but this can be achieved in distributed environment.

Hasan et al. (2018) proposed the model is controlled by Java based application as a controller which controls mobile devices and connected IOT devices. The tasks are offloaded to Aura cloud which reduce the energy consumption upto 63% of resource constrained device. Pu et al. (2016) focuses on the reduced energy consumption in offloading tasks from resource constrained devices. The model will run on different schemes for offloading they are Reciprocal, greedy and random schemes. Random scheme performs better than other schemes with 30% of energy saving. In other hand, Meurisch et al. (2017) focuses on performance of the device irrespective of the energy consumption. The idea is to approximate the performance of the complex task by sampling two tasks in the complex task. This model achieves 86% of accuracy in approximation. If the prediction of performance better at cloud, task will get offloaded to cloud.

Zhang et al. (2016) achieves 18% less energy consumption if the task get offloaded to cloud. This model neglected the latency sensitive tasks for offloading and concentrated only to computational intensive tasks. The results shows that there is no effect in performance even though there is network traffic for offloading. But Craciunescu et al. (2015) concentrated on latency sensitive application and offloading task only for time sensitive critical application with low latency. The author does not take power consumption in evaluation metrices. This model will predict the fall of network upto 90% of accuracy for decision making. Wang et al. (2016) compared fully offloaded application and the application with partial offloading. The results shows fully offloaded application consume 36% less energy compared to partially offloaded application.

2.5 Communication protocol

The RESTful API is a REST-based API designed for interaction between components. RESTful API has the ability to handle different types of calls and can able to return different data formats. Hong et al. (2018) compared RabbitMQ and RESTful API performance for microservice execution. As the result, RESTful API performs better for microservices execution with minimum latency in low network congestion. RabbitMQ method performance is not get affected if there is high network traffic.

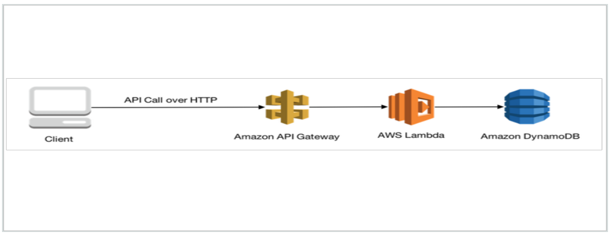


Figure 2: AWS Serverless Application Model (AWS SAM)

In this paper, AWS serverless application model is used for cloud execution. When there is high network traffic, AWS will automatically handle the scaling and provide better performance at user end. Each microservices will form lambda functions in AWS and it is integrated with AWS API gateway. As the lambda function has only relationship with particular API, it is more secure and it can be accessed through RESTful API calls. In security perspective, AWS handles DDoS attack and all request to API is made via HTTPS which enable encryption and secure connection in transit.

3 Methodology

Android application is developed with Android studio 3.6.3 IDE which is integrated with Github for version control. The application is developed with AndroidX package with Google vision library. For cloud environment, application are developed as microservices which perfectly suits to AWS lambda functions with API Gateway or Google cloud functions with Cloud endpoints. The implementation is done in AWS lambda functions with API Gateway. Lambda functions are developed in NodeJS programming language and configured with NodeJS 12.x as runtime environment and memory limit in AWS lambda function set to maximum limit (3008MB) with 10 seconds timeout.

The Complexity of the task is calculated with Mc Cabes Cyclometric method. This method will calculate the complexity of method by considering the loops and conditional statement. By assigning appropriate score to each component in code and return integer value as complexity count. If the complexity count is more than 20, it is considered as complex task if it is less than 20 will be considered as non-complex task Manukumar and Vijayalakshmi (2019). Online tool [1] is used for complexity calculation. For evaluation purpose, the android application is integrated with Firebase plugin to monitor the application activity and device activity. To analyse performance of the device locally, Android profiler as profiler tool in Android studio software. It will provide CPU activity of the device for each task and it will generate graph which can be analysed for discussion and conclusion.

4 Design Specification

In the proposed model, most widely accepted Cloudlet physical layer architecture is used. As there is no standardised architecture for Mobile cloud computing, Cloudlet architecture shows better results among others. In the Application layer architecture changes has been made to handle both latency sensitive tasks and computational intensive tasks.

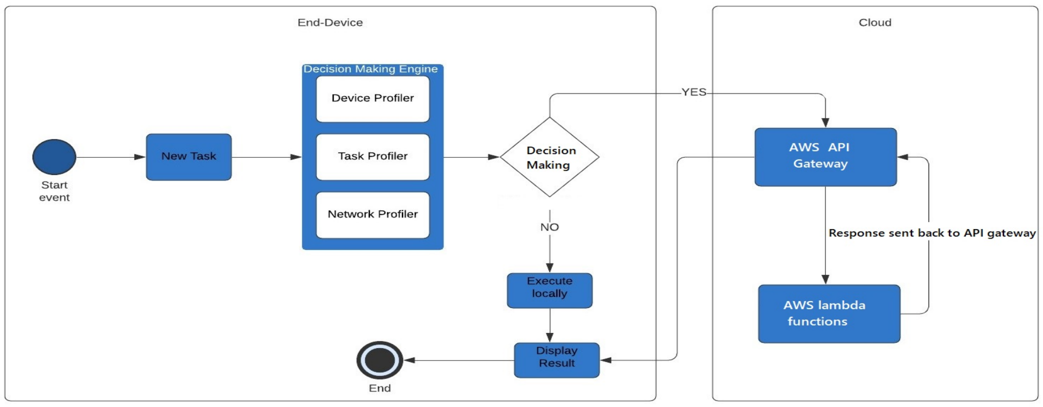


Figure 3: Process flow diagram

Android application is developed in Java programming language with latency sensitive task and computational intensive task for the evaluation of proposed approach. QR code scanner as latency sensitive task and Image editing as computational intensive task. Both task coupled in single android application.

In the proposed model, offloading technique falls under dynamic offloading which means task will get offloaded during runtime. Offloadable task will be annotated by developer in the development phase of application. As the decision making framework resides inside the application, it consists of Device profiler, Network profiler and Task profiler as separate activity and can be accessed by all other activity in the application. Device profiler will provide device status and current battery parameters, at the same time Network profiler will provide details about connection status and type of connection. Task profiler will have pre-defined data as it does not change during runtime. Each method in the task analysed offline for complexity index and stored in a Hashmap values. Task profiler will return complexity index of the method along with the type of task when requested by decision making engine.

[1]http://www.lizard.ws

Algorithm 1 Decision making algorithm

```
Result: Returns True or False
get the TaskType; get the charging status; get the usedMemory; get the totalMemory;
 get the complexity; get the batterylevel; get the threshold;
if Network connectivity then
    if userPreference==hybrid then
        if TaskType==delaySensitive then
            if connectionType==wifi then
                if isCharging==true then
                    if freeMemory greater than 50% || complexity then
                        return true
                    else
                        return false
                    end
                else
                    if batterylevel≤ threshold || freeMemory greater than 50% || complexity
                     then
                        return true
                    else
                        return false
                    end
                end
            else
                return false
            end
        else
            if isCharging==true then
                if freeMemory greater than 50% || complexity then
                    return true
                else
                    return false
                end
            else
                if batterylevel ≤ threshold || freeMemory greater than 50% || complexity
                 then
                    return true
                else
                    return false
                end
            end
        end
    else
        if userPreference==onlyCloud then
            return true
        else
            return false
        end
    end
else
    return false
end
```

Decision engine is a programming code in which with the given input along with the user preference it will decide whether the task should be executed locally or in cloud. Decision making engine works on two types of tasks

1. If the offloading task is latency-sensitive, it will check for network connection status as Wifi from Network profiler. If it is not connected to Wifi, task will get executed locally.

2. If the offloading task computational-intensive, it will request device current status and complexity of offloading task from device profiler and network status from network profiler.

When the task started, application will request decision making framework for offloading. Decision engine will check for network connection in first priority to avoid unnecessary computation in the framework if connection is not established. Decision engine will handle latency sensitive task and computation intensive task differently by considering different decision factor for each type of task.

For latency sensitive task, when the device connected to network it will further check for type of connection to which the device is connected. If the type of connection is wifi it will further proceed to next computations for decision making. In other words, latency sensitive task will not offloaded to cloud when the device is connected to Mobile data. For computational intensive task, decision making engine will check for network connection but not for type of connection, task may get offloaded even the connected to mobile data for computational intensive task.

Device profiler will provide parameters such as memory info, battery status, charging status while Network profiler will provide connection status and Task profiler will provide details about nature of the task and complexity of the task. With the available parameters decision engine will make decision for offloading. The battery threshold level set to 20% and free memory threshold is set to 50%. In the proposed model, complexity greater than 5 is considered as complex task due time frame restriction as developing complex task will require more time and effort for evaluation. If the conditions satisfies with the available parameters, the decision engine will offload task to cloud for further execution.

In the cloud side, AWS lambda functions are used which is integrated with AWS API gateway. The code has been partitioned in statically for each task and developed as lambda function. The proposed model achieves method-level granularity level in partitioning. Both QRcode scanner and Image editing functionalities are replicated in AWS lambda functions which are developed in NodeJS. These lambda functions are integrated with AWS API gateway and can be accessed through RESTful API calls.

5 Implementation

5.1 Mobile environment

5.1.1 QR code scanner

QR code scanner is latency sensitive task in which QR code reader should return result with low latency for better user experience. For local execution, google providing predefined library for all basic mobile functionalities in Androidx package. Google play services vision library is used for QR code scanner for local execution.

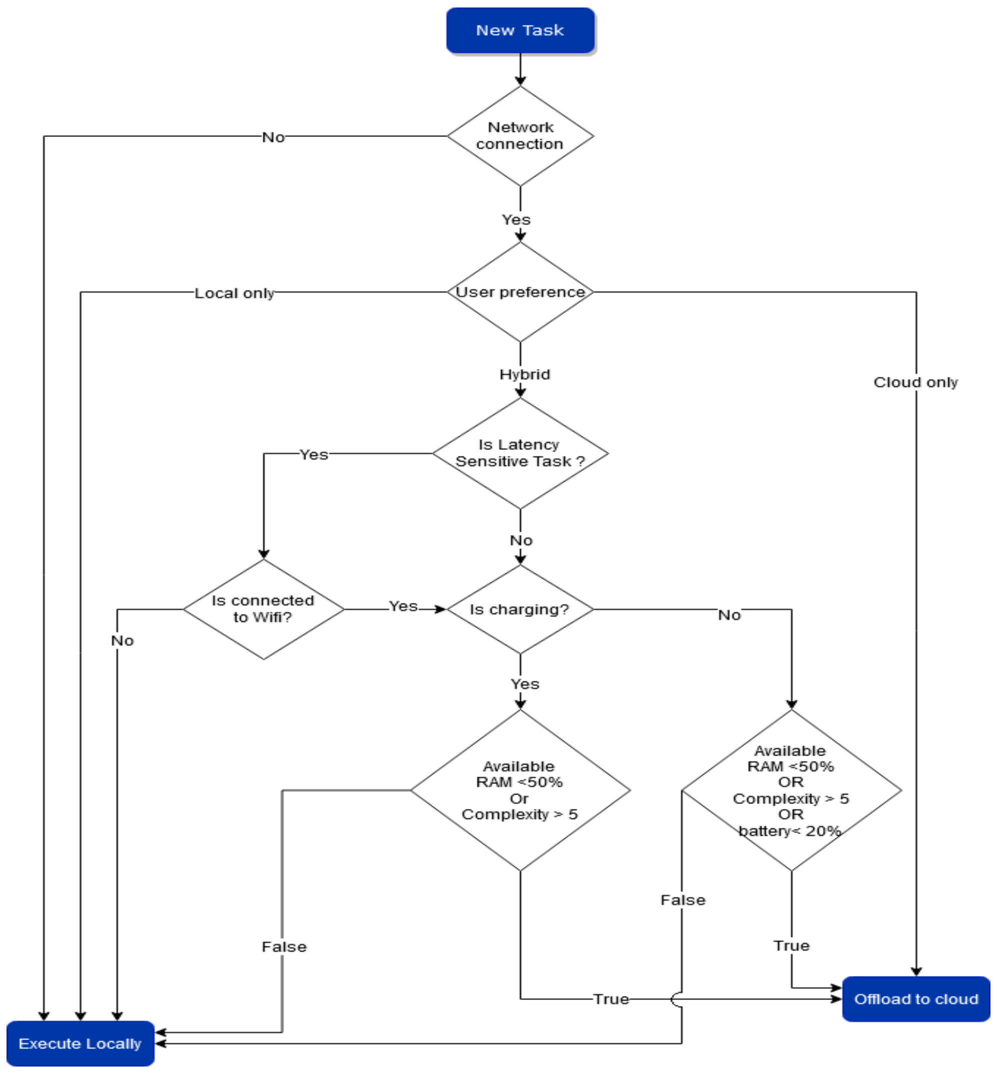


Figure 4: Decision Making Engine

Once the QR code activity is started, it request Decision making engine whether the task can be offloaded or not. In further, Decision engine will request Device profiler, Network Profiler and Task profiler for the current status of the device and it will decide accordingly and return the result to QR code activity.

If the result from decision is true, it will open the camera for image capture. The captured image will be sent to AWS lambda functions for further execution through RESTful API calls. If the result from decision engine is false, QR code activity will initialise the google play services vision library and it start to detect QR code in camera. In both cloud and local execution, the results will be available in display when the task get completed.

5.1.2 Image Editing

Image editing is computational intensive task as it involves processing of image for each pixels. For the captured picture, three processes are done as image editing. Contrast of image increased by 20% , Brightness of image increased by 20% finally color of the image is inverted.

When the Image editing activity started, it will open the camera. Once the image

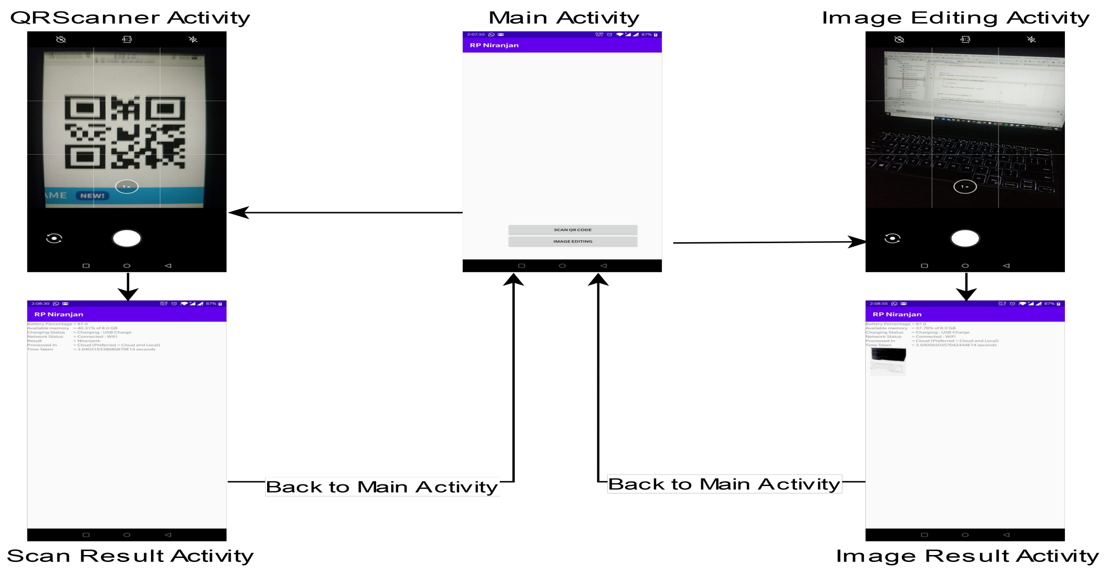


Figure 5: Android Mobile Application

is captured, it will request Decision making engine for offloading. In further decision making engine will request Device profiler, Network profiler and Task profiler for the current status of the device and it decides whether the task can be offloaded or not.

It returns the result to Image editing activity, if the result is true. it will convert image to base64 string and send to AWS lambda function in RESTful API call. Lambda function will execute the process return the result to mobile device. If the result is false, Image editing activity will start to process the image and return the result.

5.1.3 Device profiler, Network profiler and Task profiler

Device profiler will provide current status of device parameters such as Total RAM memory, Free RAM memory, Charging status, Type of charging, Battery level.In the application development, Broadcast receiver is registered for battery status change and it will broadcast to battery manager if there is any change in battery status. It will extract the type of charging whether the device connected to USB charging or AC adaptor. With the Activity Manager, device profiler will extract the memory info of the device when it is requested by decision engine.

Network profiler will monitor the network connectivity status of the device. Connectivity manager in android application will provide the connectivity status of the device and type of connection whether the device connected to Wi-Fi network or Mobile data. When the decision engine requested for Network info, Network profiler will return required network parameters to decision engine.

The proposed model is using static code partitioning and so complexity of each method is analysed offline using Java Cyclometeric method. Online tool [2] was used to calculate the complexity of each method using Java cyclometeric method which returns whole number as complexity index. In image editing, increasing brightness method and increasing contrast method has resulted with high complexity index. This obtained values for all methods in both activity were segregated and declared as HashMap values in Task profiler.

[2]http://www.lizard.ws

When decision engine requesting for complexity of task, Task profiler will provide complexity index of the particular method along with the type. Decision engine will take complexity index of the method in decision making process.

5.2 Cloud environment

AWS cloud provider has been choosed to offload the task from resource constrained device. Each task in the mobile application is converted to microservices in the cloud. AWS Serverless Application Model (AWS SAM) has been used for integration of cloud and mobile application.

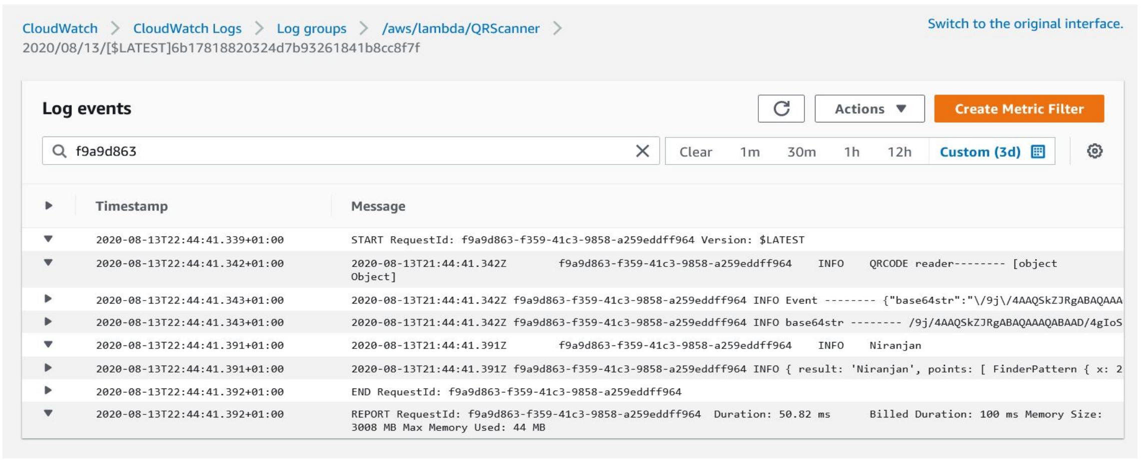
5.2.1 AWS lambda

AWS lambda is an implementation of FAAS (Function as a Service) by Amazon. In AWS lambda function, virtual server configurations and environment are provided by Amazon. But configurations and virtual server should be managed by developer in Amazon EC2 instances. Mobile application tasks are divided into microservices, so Lambda function will suit for this requirement. Each lambda function will work independently and has trust relationship only with integrated API gateway.

Each Lambda function has separate allocated RAM memory with timeout. 3008 MB memory is allocated with 10 seconds timeout for each task. So that execution will be faster with minimum failure. Compressed NodeJS application will act as lambda function in AWS. Jimp library is used for image editing and image related functionalities. qrcode-reader library is used for QR code scanning.

1. QR code Scanner

   AWS lamba function will get request with JSON object in body. The JSON object has the image as base64 string format and image is converted to Bitmap format using Jimp library. The converted image is analysed with qrcode-scanner library for the result. The result string value is converted to JSON object to send response for the request.

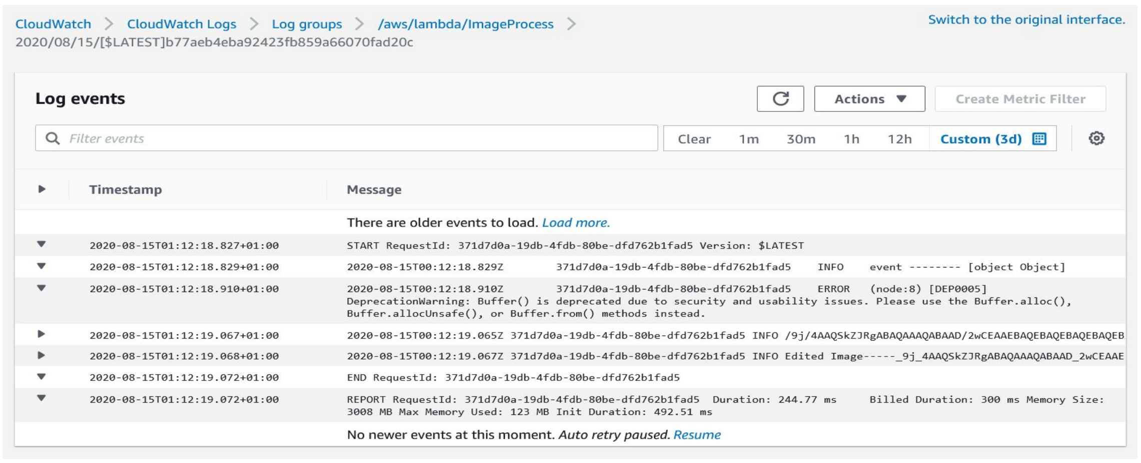

CloudWatch > CloudWatch Logs > Log groups > /aws/lambda/QRScanner >
2020/08/13/[$LATEST]6b17818820324d7b93261841b8cc8f7f
Switch to the original interface.
Log events
Actions
Create Metric Filter
f9a9d863
Clear
1m
30m
1h
12h
Custom (3d)

```
Timestamp                       Message
2020-08-13T22:44:41.339+01:00   START RequestId: f9a9d863-f359-41c3-9858-a259eddff964 Version: $LATEST
2020-08-13T22:44:41.342+01:00   2020-08-13T21:44:41.342Z        f9a9d863-f359-41c3-9858-a259eddff964    INFO    QRCODE reader-------- [object
                                Object]
2020-08-13T22:44:41.343+01:00   2020-08-13T21:44:41.342Z f9a9d863-f359-41c3-9858-a259eddff964 INFO Event -------- {"base64str":"\/9j\/4AAQSkZJRgABAQAAA
2020-08-13T22:44:41.343+01:00   2020-08-13T21:44:41.342Z f9a9d863-f359-41c3-9858-a259eddff964 INFO base64str -------- /9j/4AAQSkZJRgABAQAAAQABAAD/4gIoS
2020-08-13T22:44:41.391+01:00   2020-08-13T21:44:41.391Z        f9a9d863-f359-41c3-9858-a259eddff964    INFO    Niranjan
2020-08-13T22:44:41.391+01:00   2020-08-13T21:44:41.391Z f9a9d863-f359-41c3-9858-a259eddff964 INFO { result: 'Niranjan', points: [ FinderPattern { x: 2
2020-08-13T22:44:41.392+01:00   END RequestId: f9a9d863-f359-41c3-9858-a259eddff964
2020-08-13T22:44:41.392+01:00   REPORT RequestId: f9a9d863-f359-41c3-9858-a259eddff964  Duration: 50.82 ms      Billed Duration: 100 ms Memory Size:
                                3008 MB Max Memory Used: 44 MB
```

Figure 6: QR code scanner - AWS lambda function log for cloud execution

2. Image editing process

   As like QR code scanner, Image editing lambda function will get request with JSON object in body. Image received as base64 string format and it is converted to Bitmap format using Jimp library. The image will be processed with increasing brightness by 20% function, increasing contrast by 20% function and inverting color function. And processed image is converted again to base64 string using Jimp library. The processed image in base64 string is converted to JSON object and send back the response for the request.



CloudWatch > CloudWatch Logs > Log groups > /aws/lambda/ImageProcess >
2020/08/15/[$LATEST]b77aeb4eba92423fb859a66070fad20c

Switch to the original interface.

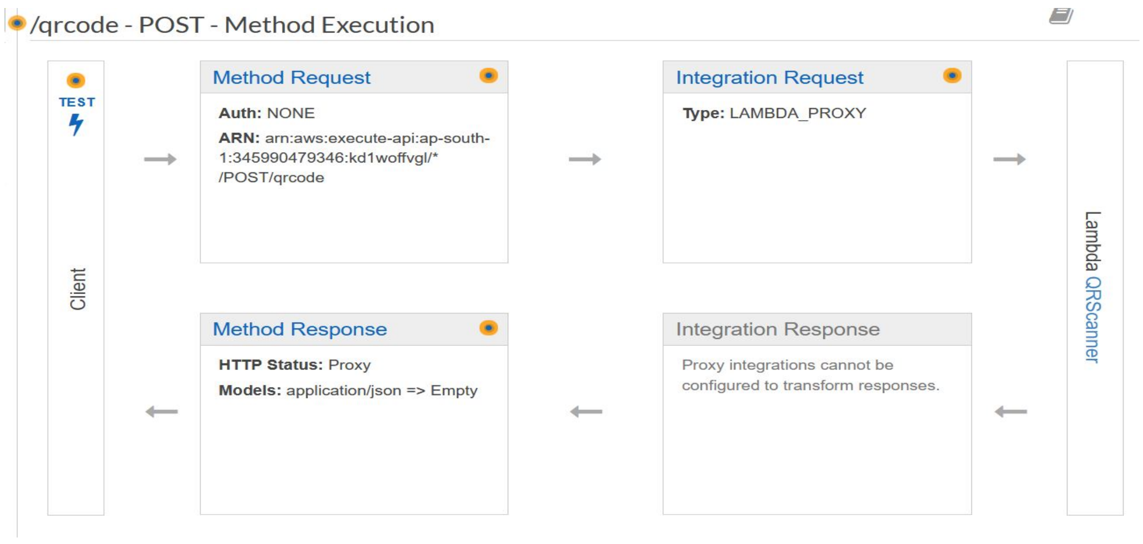
Log events

```
Timestamp                          Message
                                   There are older events to load. Load more.
2020-08-15T01:12:18.827+01:00      START RequestId: 371d7d0a-19db-4fdb-80be-dfd762b1fad5 Version: $LATEST
2020-08-15T01:12:18.829+01:00      2020-08-15T00:12:18.829Z        371d7d0a-19db-4fdb-80be-dfd762b1fad5    INFO    event -------- [object Object]
2020-08-15T01:12:18.910+01:00      2020-08-15T00:12:18.910Z        371d7d0a-19db-4fdb-80be-dfd762b1fad5    ERROR   (node:8) [DEP0005]
                                   DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.alloc(),
                                   Buffer.allocUnsafe(), or Buffer.from() methods instead.
2020-08-15T01:12:19.067+01:00      2020-08-15T00:12:19.065Z 371d7d0a-19db-4fdb-80be-dfd762b1fad5 INFO /9j/4AAQSkZJRgABAQAAAQABAAD/2wCEAAEBAQEBAQEBAQEBAQEB
2020-08-15T01:12:19.068+01:00      2020-08-15T00:12:19.067Z 371d7d0a-19db-4fdb-80be-dfd762b1fad5 INFO Edited Image-----_9j_4AAQSkZJRgABAQAAAQABAAD_2wCEAAE
2020-08-15T01:12:19.072+01:00      END RequestId: 371d7d0a-19db-4fdb-80be-dfd762b1fad5
2020-08-15T01:12:19.072+01:00      REPORT RequestId: 371d7d0a-19db-4fdb-80be-dfd762b1fad5  Duration: 244.77 ms     Billed Duration: 300 ms Memory Size:
                                   3008 MB Max Memory Used: 123 MB Init Duration: 492.51 ms
                                   No newer events at this moment. Auto retry paused. Resume
```

Figure 7: Image editing - AWS lambda function log for cloud execution

5.2.2 AWS API Gateway

AWS API gateway is integrated with lambda functions and it is configured with RESTful API. By using lambda proxy integration, there is no need to rephrase the response from lambda functions. CORS policy is enabled in API gateway. During pre-flight request from mobile device expected headers should be downloaded from response. It is enabled for security purpose.



/qrcode - POST - Method Execution

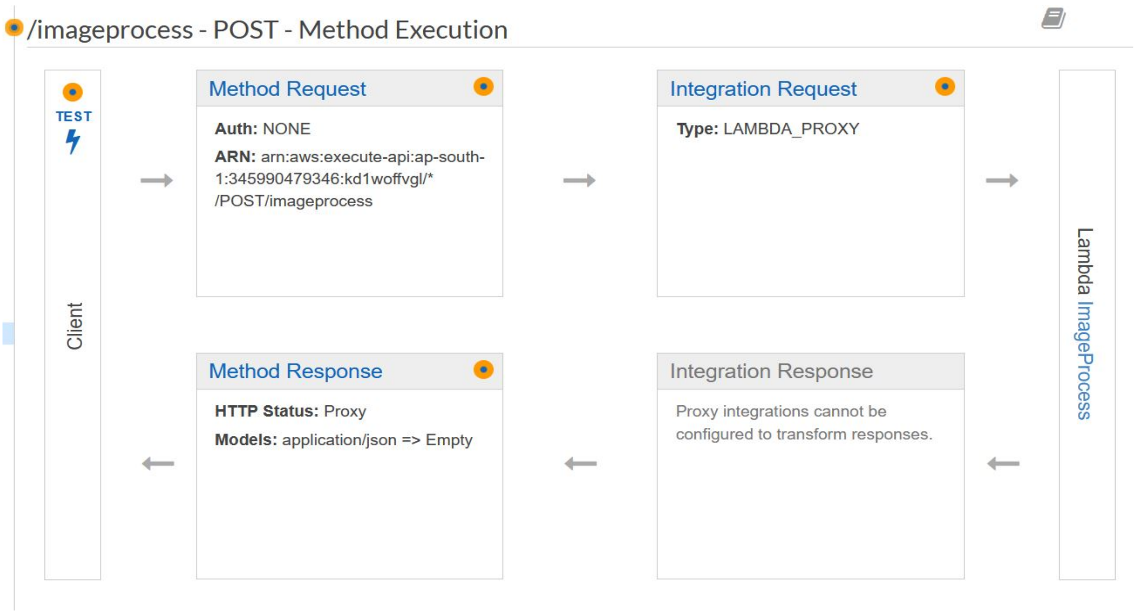
Method Request
Auth: NONE
ARN: arn:aws:execute-api:ap-south-1:345990479346:kd1woffvgl/*/POST/qrcode

Integration Request
Type: LAMBDA_PROXY

Method Response
HTTP Status: Proxy
Models: application/json => Empty

Integration Response
Proxy integrations cannot be configured to transform responses.

Figure 8: QR code scanner - AWS API Gateway POST method


/imageprocess - POST - Method Execution

TEST

Client

Method Request

Auth: NONE

ARN: arn:aws:execute-api:ap-south-1:345990479346:kd1woffvgl/*/POST/imageprocess

Integration Request

Type: LAMBDA_PROXY

Method Response

HTTP Status: Proxy

Models: application/json => Empty

Integration Response

Proxy integrations cannot be configured to transform responses.

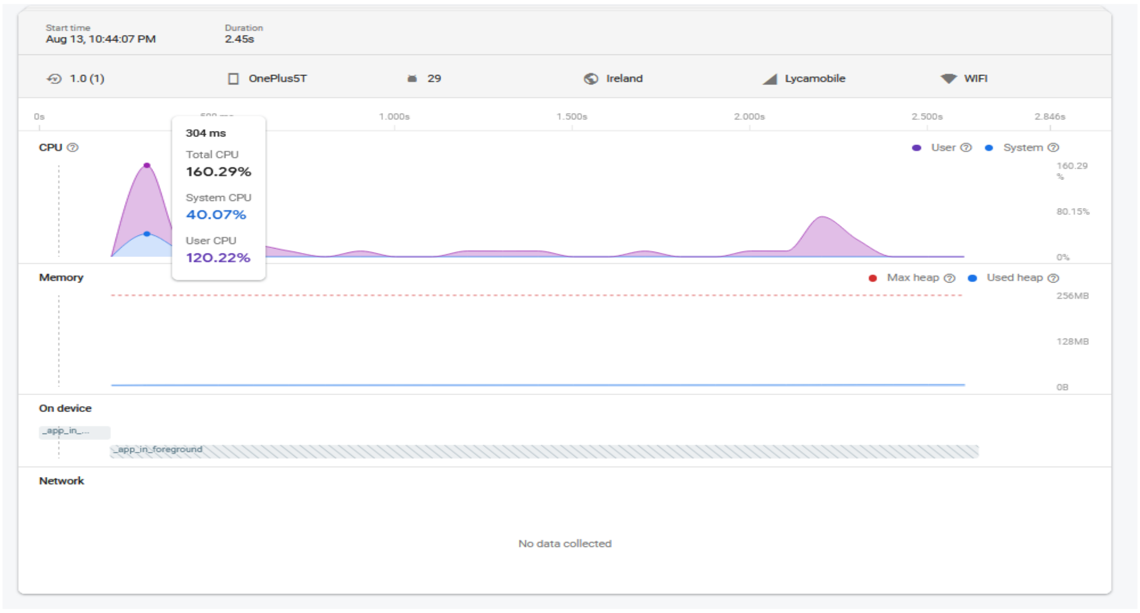
Lambda ImageProcess

Figure 9: Image editing - AWS API Gateway POST method

6 Evaluation

By offloading task to cloud, there is performance improvement in mobile device. Android studio profiler has been used along with Firebase performance plugin in application for evaluation. Experiments has been tested in ONEPLUS 5T with 8 GB RAM and 128GB ROM. It has Octa-core Kryo processor.

6.1 Experiment 1 - QR code scanner - Local execution


Start time
Aug 13, 10:44:07 PM

Duration
2.45s

1.0 (1) OnePlus5T 29 Ireland Lycamobile WIFI

304 ms
Total CPU 160.29%
System CPU 40.07%
User CPU 120.22%

No data collected

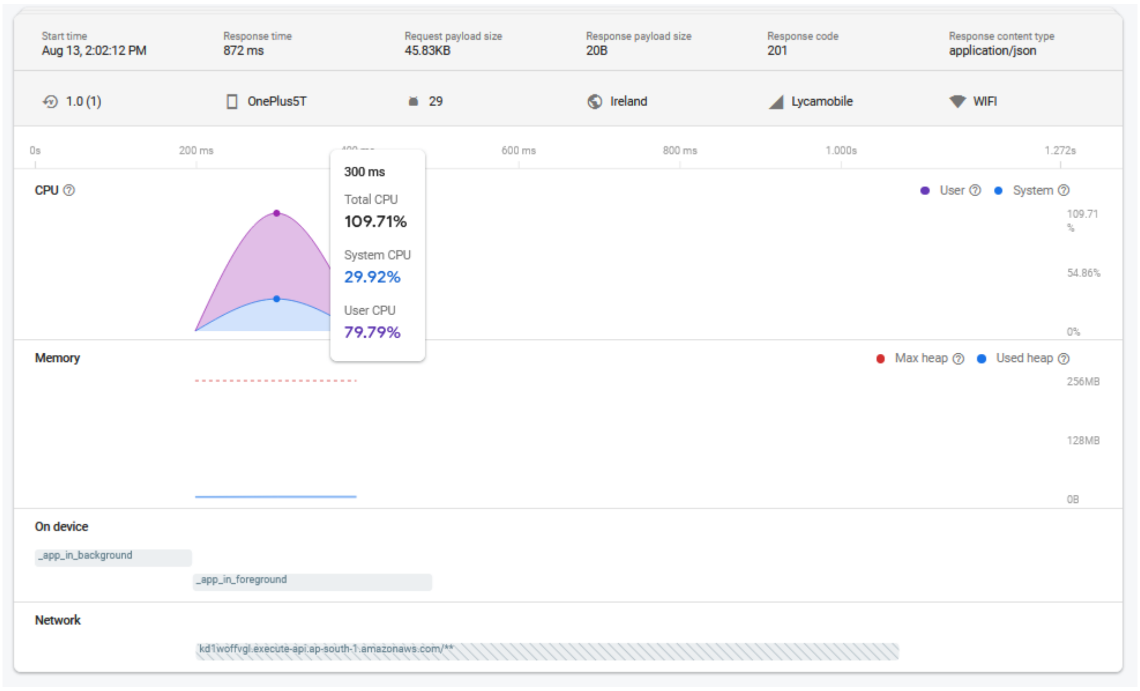
Figure 10: QR code scanner - Local execution

Android application is built using Android Studio IDE with Firebase plugin integration. In this induced scenario, with battery level 93% ,free RAM memory is 54.3% of 8.0 GB, Not charging status, Network connection as wifi. The decision has been taken to execute task in local device with the above parameters.

From the result, maximum CPU usage calculated as 160% (multiple core CPU). The User CPU usage is 120% which denotes CPU computing power used to execute application code. At the same time System CPU usage is 40% which denotes CPU computing power utilized for Android OS on behalf of application.

6.2 Experiment 2 - QR code scanner - Cloud execution

In this scenario, the battery percentage is 100%, free RAM memory is 30.43% of 8.0 GB, Charging in USB, Network connection as wifi. With the given condition, the task will get offloaded to cloud to get back the result.



Start time	Response time	Request payload size	Response payload size	Response code	Response content type
Aug 13, 2:02:12 PM	872 ms	45.83KB	20B	201	application/json
1.0 (1)	OnePlus5T	29	Ireland	Lycamobile	WIFI

300 ms
Total CPU 109.71%
System CPU 29.92%
User CPU 79.79%

On device: _app_in_background, _app_in_foreground

Network: kd1woffvgl.execute-api.ap-south-1.amazonaws.com/**

Figure 11: QR code scanner - Cloud execution

From the result, total CPU usage is 109% which includes User CPU utilization 80% and System CPU utilization 30% which is less than the locally executed task usage.

6.3 Experiment 3 - Image editing - Local execution

For image editing task, the battery level is set to 100% with charging status, free RAM memory is 30.33% of 8.0 GB, without Network connection. With the above parameters, the decision engine will decide to execute task locally.

From the figure 12, when the image editing executed locally, the CPU usage went upto 290% which proves that the task is computational intensive. Device will consume energy proportional to the usage of the CPU.

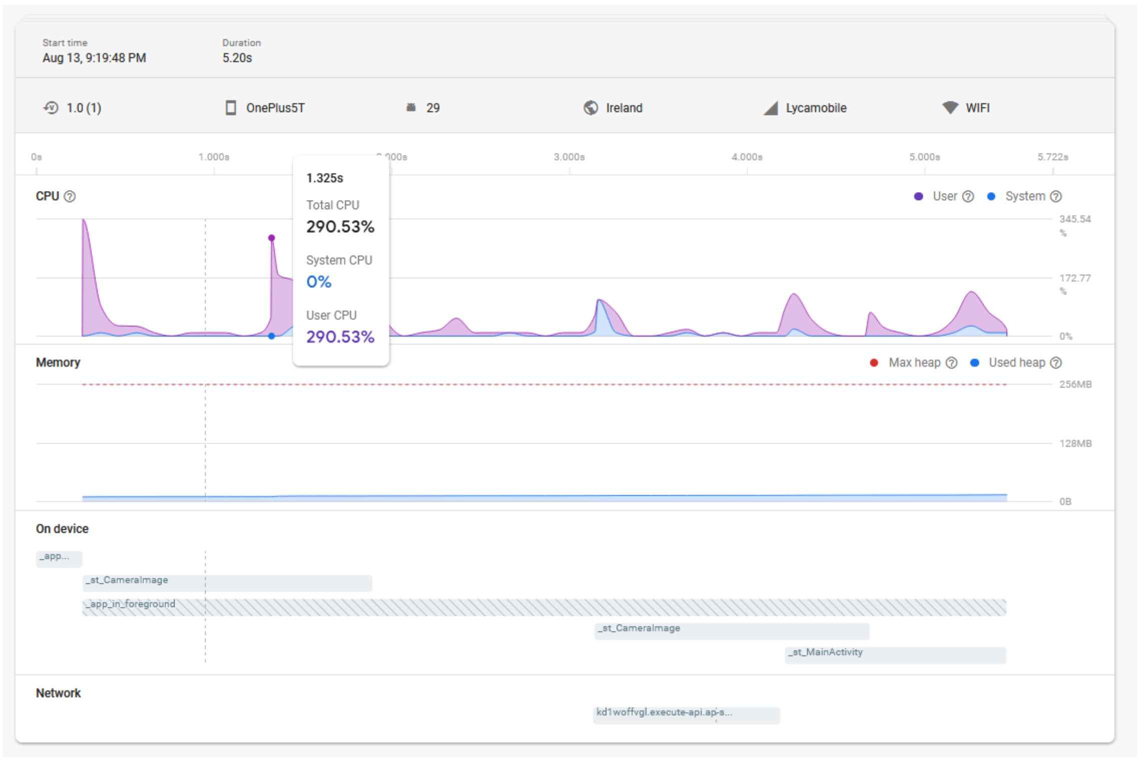


Figure 12: Image editing - Local and Cloud execution

6.4 Experiment 4 Image editing - Cloud

For cloud execution of image processing, the battery level is set to 100% with charging status, free RAM memory is 30.33% of 8.0 GB, Network connection as wifi. The decision has made the decision to offload the task with above parameters.

From the figure 12, when the image editing is executed in cloud, the CPU usage can be seen in graph which is less than 172% and it utilize low CPU usage when compared to local execution.

6.5 Discussion

The proposed novel Decision making framework which handles both Latency sensitive and Computational intensive task shows better results with improvement in device performance. Jaskaniec (2019) formulated decision making framework for offloading generic task and device performance is not evaluated with CPU activity but the proposed model will consider different decision factor for each type of task and the task will get offloaded to AWS Lambda functions for execution. Stable network bandwidth is needed for offloading Latency sensitive task and considering network connection type as a major part in decision making for Latency sensitive task to avoid latency issues. While for computational intensive task, any type of network connection will favours offloading. The proposed decision making framework can be integrated to any existing application by adding some java classes along with their original source code. On the cloud side, the decision making framework not cloud provider dependent. Cloud provider can be changed without changing the source code of application. The decision framework does not take battery energy consumption metrics and network bandwidth strength. By incorporating machine learning algorithm in decision making framework for accurate prediction for offloading will be the future scope of this paper. As this paper proposed framework for Android device, the research can be extended to iOS device in future work.

7 Conclusion and Future Work

Many framework and offloading techniques has been proposed in mobile cloud computing which focuses on different metrices. The offloading technique mostly focuses either on the latency issues for latency-sensitive application or a strategy to reduce energy consumption and computation overhead. But, the proposed approach will handle both latency sensitive task and computation intensive task by considering different decision factor to achieve better performance of local device. With reduced usage of CPU, it performs better than other existing techniques by considering different decision factor and device context. The Future work of the proposal will be in Network profiler. Network parameters such as signal strength and bandwidth have to be considered for decision making with the implication of machine learning algorithm. Thus, the decision making framework accuracy will get increased if machine learning algorithm introduced.

References

Aazam, M., St-Hilaire, M. and Huh, E. (2016). Towards media intercloud standardization evaluating impact of cloud storage heterogeneity, *Journal of Grid Computing* **abs/1602.06246**. JCR Impact Factor: 2.800 (2018).
URL: *http://arxiv.org/abs/1602.06246*

Abdo, J. B. and Demerjian, J. (2017). Evaluation of mobile cloud architectures, *Pervasive and Mobile Computing* **39**: 284–303. JCR Impact Factor: 2.974 (2018).
URL: *https://doi.org/10.1016/j.pmcj.2016.12.003*

Benedetto, J. I., González, L. A., Sanabria, P., Neyem, A. and Navón, J. (2019). Towards a practical framework for code offloading in the internet of things, *Future Generation Computer Systems-The International Journal of eScience* **92**: 424–437. JCR Impact Factor: 4.639 (2018).
URL: *https://doi.org/10.1016/j.future.2018.09.056*

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R. N., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A. S., Toosi, A. N., Rodriguez, M. A., Llorente, I. M., di Vimercati, S. D. C., Samarati, P., Milojicic, D. S., Varela, C. A., Bahsoon, R., de Assunção, M. D., Rana, O., Zhou, W., Jin, H., Gentzsch, W., Zomaya, A. Y. and Shen, H. (2019). A manifesto for future generation cloud computing: Research directions for the next decade, *ACM Comput. Surv.* **51**(5): 105:1–105:38. JCR Impact Factor: 5.550 (2018).
URL: *https://doi.org/10.1145/3241737*

Chun, B., Ihm, S., Maniatis, P., Naik, M. and Patti, A. (2011). Clonecloud: elastic execution between mobile device and cloud, *in* C. M. Kirsch and G. Heiser (eds), *European Conference on Computer Systems, Proceedings of the Sixth European conference on Computer systems, EuroSys 2011, Salzburg, Austria, April 10-13, 2011*, ACM, pp. 301–314. CORE Ranking: ”A”.
URL: *https://doi.org/10.1145/1966445.1966473*

Craciunescu, R., Mihovska, A., Mihaylov, M., Kyriazakos, S., Prasad, R. and Halunga, S. (2015). Implementation of fog computing for reliable e-health applications, *2015 49th Asilomar Conference on Signals, Systems and Computers*, pp. 459–463. CORE Ranking: "C".

Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R. and Bahl, P. (2010). MAUI: making smartphones last longer with code offload, *in* S. Banerjee, S. Keshav and A. Wolman (eds), *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010), San Francisco, California, USA, June 15-18, 2010*, ACM, pp. 49–62. CORE Ranking: "B".
URL: *https://doi.org/10.1145/1814433.1814441*

Elgazar, A., Harras, K. A., Aazam, M. and Mtibaa, A. (2018). Towards intelligent edge storage management: Determining and predicting mobile file popularity, *6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2018, Bamberg, Germany, March 26-29, 2018*, IEEE Computer Society, pp. 23–28.
URL: *https://doi.org/10.1109/MobileCloud.2018.00012*

Fricker, C., Guillemin, F., Robert, P. and Thompson, G. (2016). Analysis of an offloading scheme for data centers in the framework of fog computing, *TOMPECS* **1**(4): 16:1–16:18.
URL: *https://doi.org/10.1145/2950047*

Hasan, R., Hossain, M. M. and Khan, R. (2018). Aura: An incentive-driven ad-hoc iot cloud framework for proximal mobile computation offloading, *Future Generation Computer Systems-The International Journal of eScience* **86**: 821–835. JCR Impact Factor: 4.639 (2018).
URL: *https://doi.org/10.1016/j.future.2017.11.024*

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

Huang, C., Lu, R. and Choo, K. R. (2017). Vehicular fog computing: Architecture, use case, and security and forensic challenges, *IEEE Communications Magazine* **55**(11): 105–111. JCR Impact Factor: 9.270 (2018).
URL: *https://doi.org/10.1109/MCOM.2017.1700322*

Jaskaniec, A. (2019). *Mobile task offloading based on bandwidth and battery availability*, Master's thesis, Dublin, National College of Ireland.
URL: *http://trap.ncirl.ie/3842/*

Kemp, R., Palmer, N., Kielmann, T. and Bal, H. E. (2010). Cuckoo: A computation offloading framework for smartphones, *in* M. L. Griss and G. Yang (eds), *Mobile Computing, Applications, and Services - Second International ICST Conference, MobiCASE 2010, Santa Clara, CA, USA, October 25-28, 2010, Revised Selected Papers*, Vol. 76 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer, pp. 59–79. CORE Ranking: "A".
URL: *https://doi.org/10.1007/978-3-642-29336-8_4*

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, *in* A. G. Greenberg and K. Sohraby (eds), *Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012*, IEEE, pp. 945–953. CORE Ranking: "A*".
URL: *https://doi.org/10.1109/INFCOM.2012.6195845*

Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R. and Qureshi, A. (2015). Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions, *Journal of Network And Computer Applications* **48**: 99–117. JCR Impact Factor: 3.991 (2018).
URL: *https://doi.org/10.1016/j.jnca.2014.09.009*

Manukumar, S. T. and Vijayalakshmi, M. (2019). A novel multi-objective efficient offloading decision framework in cloud computing for mobile computing applications, *Wireless Personal Communications* **107**(4): 1625–1642. JCR Impact Factor: 1.200 (2018).
URL: *https://doi.org/10.1007/s11277-019-06348-4*

Masip-Bruin, X., Marín-Tordera, E., Tashakor, G., Jukan, A. and Ren, G. (2016). Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems, *IEEE Wireless Communications* **23**(5): 120–128. JCR Impact Factor: 9.202 (2018).

Meurisch, C., Gedeon, J., Nguyen, T. A. B., Kaup, F. and Mühlhäuser, M. (2017). Decision support for computational offloading by probing unknown services, *26th International Conference on Computer Communication and Networks, ICCCN 2017, Vancouver, BC, Canada, July 31 - Aug. 3, 2017*, IEEE, pp. 1–9. CORE Ranking: "A".
URL: *https://doi.org/10.1109/ICCCN.2017.8038406*

Montella, R., Kosta, S., Oro, D., Vera, J., Fernández, C., Palmieri, C., Luccio, D. D., Giunta, G., Lapegna, M. and Laccetti, G. (2017). Accelerating linux and android applications on low-power devices through remote GPGPU offloading, *Concurrency and Computation-Practice Experience* **29**(24). JCR Impact Factor: 1.114 (2018).
URL: *https://doi.org/10.1002/cpe.4286*

Noor, T. H., Zeadally, S., Alfazi, A. and Sheng, Q. Z. (2018). Mobile cloud computing: Challenges and future research directions, *J. Network and Computer Applications* **115**: 70–85. JCR Impact Factor: 3.991 (2018).
URL: *https://doi.org/10.1016/j.jnca.2018.04.018*

Osanaiye, O. A., Chen, S., Yan, Z., Lu, R., Choo, K. R. and Dlodlo, M. E. (2017). From cloud to fog computing: A review and a conceptual live VM migration framework, *IEEE Access* **5**: 8284–8300. JCR Impact Factor: 3.557 (2018).
URL: *https://doi.org/10.1109/ACCESS.2017.2692960*

Pu, L., Chen, X., Xu, J. and Fu, X. (2016). D2D fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted D2D collaboration, *IEEE Journal on Selected Areas in Communications* **34**(12): 3887–3901. JCR Impact Factor: 7.172 (2018).
URL: *https://doi.org/10.1109/JSAC.2016.2624118*

Wang, Y., Sheng, M., Wang, X., Wang, L. and Li, J. (2016). Mobile-edge computing: Partial computation offloading using dynamic voltage scaling, *IEEE Trans. Communications* **64**(10): 4268–4282. JCR Impact Factor: 4.671 (2018).
URL: *https://doi.org/10.1109/TCOMM.2016.2599530*

Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S. and Zhang, Y. (2016). Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks, *IEEE Access* **4**: 5896–5907. JCR Impact Factor: 3.557 (2018).
URL: *https://doi.org/10.1109/ACCESS.2016.2597169*

Zhao, X., Zhao, L. and Liang, K. (2016). An energy consumption oriented offloading algorithm for fog computing, *in* J. Lee and S. Pack (eds), *Quality, Reliability, Security and Robustness in Heterogeneous Networks - 12th International Conference, QShine 2016, Seoul, Korea, July 7-8, 2016, Proceedings*, Vol. 199 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer, pp. 293–301. CORE Ranking: "B".
URL: *https://doi.org/10.1007/978-3-319-60717-7_29*