International Journal For Technological Research In Engineering
Volume 4, Issue 12, August-2017 *ISSN (Online): 2347 - 4718*

A SELECTIVE OFFLOADING FRAMEWORK FOR MOBILE CLOUD COMPUTING

Soumya Nair[1], Kompal Paliwal[2], Mitali Manjarekar[3], Ketki Kokate[4]
Department of Information Technology, Cummins College of Engineering for Women, Pune.

Abstract: In the recent years cloud computing has emerged as a technology which allows users to store and compute over the network and pay only for what they use. Exploiting this technology we can achieve better performance in the case of mobile devices with limited processing power and battery life, for complex and sophisticated mobile applications which are exploding in their popularity. This paper presents an optimization framework where selective offloading is being performed on a computationally expensive application. The application is being partitioned using JADE (Java Application Development Environment) and then offloaded to the cloud servers accordingly. We use Google Firebase Cloud Functions for heavy computations. This process reduces the energy consumption and execution overhead of the mobile device. We have also discussed the advantages of using this framework on the basis of various performance parameters like memory, data, CPU usage and battery consumption. Experimenting this framework with real-world mobile applications demonstrates the superiority of this approach over monolithic execution of resource-intensive applications on mobile devices.
Index Terms: Cloud Computing, Optimization Framework, Selective Offloading

I. INTRODUCTION

Over the last decade, the use of smartphones has increased exponentially. The number of smartphone users in India in 2017 is reported to be around 300 million. 54% of the world's population has been predicted to own a smartphone by 2018. Today we are at a point that in less than 2 years, a smartphone would be the only computer we owned. These mobile phones have evolved in order to imitate the services provided by the highly advanced desktop computers. Smartphones today are easily replacing the formerly popular desktop PCs with their comparable features like 6+ gigabytes of RAM, powerful graphics hardware and multiple processors. Powerful computing capabilities have made these smartphones increasingly popular. With the advancement in the technology of smartphones, users heavily rely on them for their day-to- day tasks such as e-banking, healthcare applications etc. Emails can be handled from anywhere and GPS can be used for tracking any location. It has replaced so many devices like dictionaries, scanners, kindle etc. and shall replace many more devices like set-top boxes, IDs and many more. Nearly all tasks that can be performed on mid-range desktop computers can be performed on smartphones and tablets. Although smartphones today show nearly similar performances as that of desktop computers, they will still face limitations in terms of resources to fulfill its requirement of being light and handy. The demand for the smartphones to be sleek and handy is really high these days which poses as a setback in terms of its resources. Also using these devices for highly computationally intensive real-time applications like image processing could put unnecessary load on the processor and drain the battery of the device. Inspite of all other features being comparable to the desktop, battery limitations of the smartphones still persist. The mobile industry's foremost challenge today is tackling the hurdles that come with massive energy requirements of the phone. Also with the rapid advancement in technology the newest smartphone model turns obsolete within the course of a few months, meanwhile the applications are progressively getting complicated. In such a situation buying new devices every three months is not a cost-effective solution. At the same time the users must get access to the new applications and should be able to avail them without having to discard their old phones. Strategic use should be made of the hardware in order to utilize it more efficiently. Here is where cloud computing comes to the rescue. Cloud computing enables us to remove the load off the device and carry out the computation on remote well equipped machines. Linking smartphones with cloud helps in building a more sophisticated software. Computational offloading to more powerful servers is the solution to overcome these constraints and provide these devices with the resources they need to achieve complex tasks in real-time. Cloud computing needs to manage the resources on the basis of their availability and their demand. It provides us this opportunity to execute our applications on servers instead of running them locally and favors us to overcome the handsets limitation of limited resources to a great extent. These processor intensive tasks can be offloaded to the cloud by leveraging infrastructure such as Google's Firebase. The rest of this paper presents the design, implementation and evaluation of this framework. Our framework achieves the above mentioned objectives by providing a novel execution offloading infrastructure to make it easy for developers to exploit the framework with minimal modification of the existing code. We provide an easy to adapt and cost effective solution not just for the convenience of the developers but also for the users.

II. PREVIOUS WORK

Overview

Mobile-cloud computing involves the collaboration between mobile and cloud computing. In this paper we have proposed an optimization framework for mobile cloud computing that dynamically decides which agent-based application partition must be offloaded to the cloud and which one must be processed on the mobile device itself. Mobile cloud

computing has the potential to bridge the gap between resource requirements and availability.

Traditional Approach and its Setbacks

Cloud computing is used by many mobile applications today that are mostly involved in an inflexible split of computation between the mobile and cloud platforms, following the client-server paradigm with hard- coded interactions with the server preventing applications from adapting to conditions such as high network latency, resulting in poor performance when cloud resources are preferred over computation on the device. Optimal partitioning of the mobile application components between the mobile and cloud platforms based on runtime conditions. Frameworks with various partitioning and optimization techniques have been proposed recently. Efforts in computation offloading have a long history; research in mobile-cloud computing is still at its infancy. CloneCloud and MAUI partition applications using a framework that combines static program analysis with dynamic program profiling, thus optimizing the execution time and energy consumption on the mobile device using an optimization solver. A copy of the whole application code/virtual machine at the remote execution site is required which becomes a drawback, this makes the application code vulnerable to analysis by malicious parties on the same platform and imposes a strict requirement for public cloud machines

New Approach with Mobile Agents

No pre-requisite requirements on the cloud platform other than providing isolated execution containers is needed, autonomous agent-based application partitions alleviate the management burden of offloaded code by the mobile platform. A Mobile Agent, namely, is a type of software agent, with the feature of autonomy, social ability, learning, and most significantly, mobility. More specifically, a mobile agent is a process that can transport its state from one environment to another, with its data intact, and be capable of performing appropriately in the new environment. Mobile agents decide when and where to move. Movement is often evolved from RPC methods. Just as a user directs an Internet browser to visit a website (the browser merely downloads a copy of the site or one version of it in the case of dynamic web sites), similarly, a mobile agent accomplishes a move through data duplication. When a mobile agent decides to move, it saves its own state (process image), transports this saved state to the new host, and resumes execution from the saved state. A mobile agent is a specific form of mobile code, within the field of code mobility. However, in contrast to the Remote evaluation and Code on demand programming paradigms, mobile agents are active in that they can choose to migrate between computers at any time during their execution. This makes them a powerful tool for implementing distributed applications in a computer network. There are two types of mobile agent. The classification is based on their migration path.

- Mobile agents with predefined path: Have static migration path
- Free roaming mobile agent: Have dynamic migration path. Depending up on the present network condition the mobile agent chooses its path.

Some advantages which mobile agents have over conventional agents are:

1. Computation bundles - converts computational client/server round trips to relocatable data bundles, reducing network load.
2. Parallel processing -asynchronous execution on multiple heterogeneous network hosts
3. Dynamic adaptation - actions are dependent on the state of the host environment
4. Tolerant to network faults - able to operate without an active connection between client and server
5. Flexible maintenance - to change an agent's actions, only the source (rather than the computation hosts) must be updated
6. Bandwidth conversion which is conversion the bandwidth one host to another host.
7. Reduction in compilation time.

Selective Offloading

The transfer of certain computing tasks to an external platform, such as a cluster, grid, or a cloud is computation offloading. It may be necessary due to hardware limitations of a computer system handling a particular task on its own. SOME architecture is an offloading system for mobile applications, in an effort to reduce the computational cost of mobile devices. The limited processing power and battery lifetime of mobile phones hinder the possible execution of computationally intensive applications like content-based video analysis or 3D modeling. This problem can be addressed by offloading of computationally intensive application parts from the mobile platform into a remote cloud infrastructure or nearby idle computers.

JADE-Java Agent Development Framework

JADE is a distributed agent's platform, which has a container for each host where you are running the agents. Agents live on top of a Platform that provides them with basic services such as message delivery. A platform is composed of one or more Containers. Containers can be executed on different hosts thus achieving a distributed platform. Each container can contain zero or more agents. Additionally the platform has various debugging tools, mobility of code and content agents, the possibility of parallel execution of the behavior of agents, as well as support for the definition of languages and ontologies. Agents execute tasks and interact by exchanging messages.

III. SYSTEM ARCHITECTURE

The components involved in the architecture of the selective offloading framework are JADE, mobile device and cloud service directory which is part of the Google Firebase system. Figure 1 shows the higher level architecture of the framework. Set of agent- based application partitions that are offloadable to the cloud for execution (P2 and P3 in the figure) are part of the architecture, there are a set of native application components that are always executed on the

device due to constraints such as accessing native sensors of the device or providing the user interface of the application (P1 in the figure). Partitioning of the application is done statically right now. During the offline application partitioning process, program partitions that are not computationally intensive are set as an offloadable component.

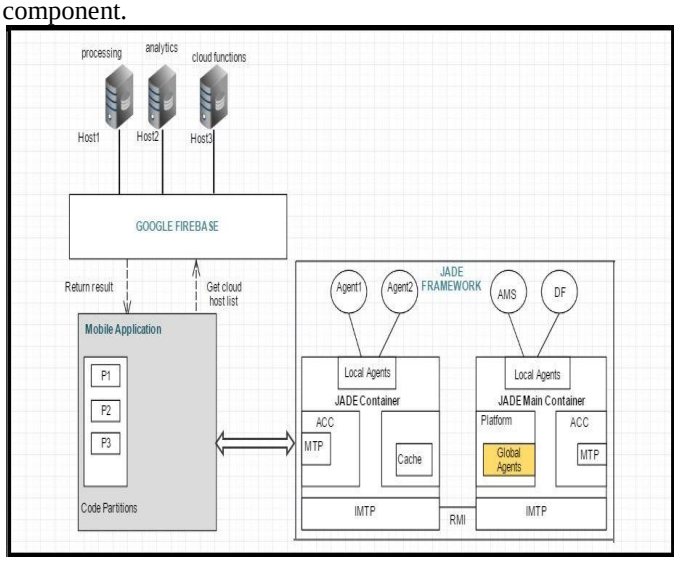

Fig 3.1

When a mobile application is launched, the framework contacts the Google Firebase to get a list of available machine instances in the cloud and selects the instance(s) with the highest communication speed with the mobile device and the highest computing power. After this step executes, offloading decisions for the agent-based partitions is created. If the execution plan requires offloading a particular application partition, a bridge is formed between the caller of that partition and the cloud host selected by the Google Firebase, through which the offloaded partition migrates to the container in the host, carrying along its input parameters. Upon migration, the partition starts executing and communicates its output data to the caller through the same bridge.

The main components of the proposed framework are described below:

MODULES

1. JADE

An application based on JADE is made of a set of components called Agents each one having a unique name. Agents execute tasks and interact by exchanging messages. Agents live on top of a Platform that provides them with basic services such as message delivery. A platform is composed of one or more Containers. Containers can be executed on different hosts thus achieving a distributed platform. Each container can contain zero or more agents. Each platform must have a parent container that has two special agents called AMS and DF. The DF (Directory Facilitator) provides a directory which announces which agents are available on the platform. The AMS (Agent Management System) controls the platform. Is the only one who can create and destroy other agents, destroy containers and stop the platform. To access the AMS Service an agent is created which automatically runs the register method of the AMS by default before executing the method setup from the new agent. When an agent is destroyed it executes its takedown() method by default and automatically calls the deregister method of the AMS. An application based on JADE is made of a set of components called Agents each one having a unique name. Agents execute tasks and interact by exchanging messages. Agents live on top of a Platform that provides them with basic services such as message delivery. A platform is composed of one or more Containers. Containers can be executed on different hosts thus achieving a distributed platform. Each container can contain zero or more agents. JADE was initially developed by Telecom Italia Lab. This sector is the R and D branch of Telecom Italia Group which is responsible for promoting technological innovation. Telecom Italia conceived and promoted JADE by basing it in 2000. In March 2003 Motorola and Telecom Italia create the JADE Governing Board with the objective of promoting the development and adoption of JADE in the mobile telecommunications industry as middleware based. That organization (JADE Governing Board) accepts to any company and / or organization interested in the commercial use and exploitation of JADE to commit to its development and promotion.

2. Firebase

Firebase is a mobile and web application platform with tools and infrastructure designed to help developers build high-quality apps. Firebase is made up of complementary features that developers can mix-and-match to fit their needs. Most Firebase features are free forever, for any scale. User doesn't have to worry about scaling the server code or provisioning extra capacity Firebase takes care of that. Cloud Functions for Firebase lets you create functions that are triggered by Firebase products, such as changes to data in the Real-time Database, uploads to Cloud Storage, new user sign ups via Authentication, and conversion events in Analytics. The ability to extend and connect Firebase features using Cloud Functions makes Firebase more powerful, allowing you to do even more with your app. Cloud Functions is a hosted, private, and scalable Node.js environment where you can run JavaScript code. Firebase SDK for Cloud Functions integrates the Firebase platform by letting you write code that responds to events and invokes functionality exposed by other Firebase features. In many cases, application logic is best controlled on the server in order to avoid tampering on the client side. Cloud Functions are fully insulated from the client so you can be sure they are private and secure and can't be reverse engineered. It requires zero maintenance. One can deploy the code to the servers with one command from the command line. On doing that Firebase automatically scales up computing resources to match the usage patterns of your users. Credentials, server configuration, provisioning new servers, or decommissioning old ones is handled automatically by Firebase. It helps in keeping the logic private and secure. In many cases, developers prefer to control application logic on the server to

avoid tampering on the client side. Also, sometimes it's not desirable to allow that code to be reverse engineered. Cloud Functions is fully insulated from the client, so you can be sure it is private and always does exactly what you want. As a real-time, scalable backend, Firebase provide the tools you need to quickly build rich, collaborative applications that can serve the users.

Lifecycle of a Cloud Function for Firebase

- The developer writes code for a new Cloud Function, selecting an event provider (such as Real-time Database), and defining the conditions under which the Cloud Function should execute.
- The developer deploys the Cloud Function, and Firebase connects it to the selected event provider.
- When the event provider generates an event that matches the Cloud Function's conditions, the code is invoked.
- If the Cloud Function is busy handling many events, Google creates more instances to handle work faster. If the Cloud Function is idle, instances are cleaned up.
- When the developer updates the Cloud Function by deploying updated code, all instances for the old version are cleaned up and replaced by new instances.
- When a developer deletes the Cloud Function, all instances are cleaned up and the connection between the Cloud Function and the event provider is removed.

IV. WORKING OF THE SYSTEM

After you write and deploy a Cloud Function, Google's servers begin to manage the function immediately, listening for events and running the function when it is triggered. As the load increases or decreases, Google responds by rapidly scaling the number of virtual server instances needed to run your function. In this section the step by step working of the complete system as a unit has been described:

- The partitioned application code is given to JADE.
- JADE wraps it along with the information required for the code's execution to create a mobile agent.
- The application is then launched
- Once the application is launched, agents offload the functions to cloud i.e.
- Google Firebase.
- The computation of the function is done on the cloud host, the computed result is returned back and stored on the cloud.
- The results are then displayed on the interface where the user can access it.

V. EXPERIMENTAL ANALYSIS

An experiment was performed on a locally developed mobile application called Steganography App. The application used the concept of steganography which is concealing text within an image. The text is embedded with the image and on reception of the image, does not appear with it but can be extracted. The experiment was used to evaluate performance of the framework in terms of application memory usage, processor usage and battery usage. An emulator of Google Nexus 4G device running Android 4.4 operating system was used to run the mobile application and Google Firebase instances were used as cloud hosts. For the offloaded execution, a moderate speed Wi-Fi connection was used to send/receive data from the cloud servers. The application user requires to have a Google Firebase account and needs to sign- in to get access to the images. To measure the energy consumed by the applications due to CPU and Wi-Fi utilization, we used PCloudy, an online testing tool for mobile applications.

Figures 4.1 and 4.2 show sample ways to upload an image and create mobile agents.

```
public void onClick(View view)
    {
        imageContainer.setDrawingCacheEnabled(true);
        imageContainer.buildDrawingCache();
        Bitmap bitmap= imageContainer.getDrawingCache();
        ByteArrayOutputStream baos= new ByteArrayOutputStream();
        bitmap.compress(Bitmap.CompressFormat.PNG, 100, baos);
        imageContainer.setDrawingCacheEnabled(false);
        byte[] data= baos.toByteArray();

        String path= "drawable/"+ UUID.randomUUID()+".png";
        StorageReference firememeRef = storage.getReference(path);

        StorageMetadata metadata=new StorageMetadata.Builder()
                .setCustomMetadata("text", overlayText.getText().toString())
                .build();

        progressBar.setVisibility(View.VISIBLE);
        uploadButton.setEnabled(false);

        UploadTask uploadTask = firememeRef.putBytes(data, metadata);
        uploadTask.addOnSuccessListener(MainActivity.this, new OnSuccessListener<UploadTask.TaskSnapshot>() {
            @Override
            public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                progressBar.setVisibility(View.GONE);
                uploadButton.setEnabled(true);

                Uri url=taskSnapshot.getDownloadUrl();
                downloadUrl.setText(url.toString());
                downloadUrl.setVisibility(View.VISIBLE);
            }
        });
    }
```

Fig 4.1

```
private void createAgent(String name,String news,String className) {
            if (mainContainerHandler != null) {
                Object[] args = new Object[1];
                args[0] =news;
                mainContainerHandler.createNewAgent(name, className,
                        args, new RuntimeCallback<AgentHandler>() {
                            @Override
                            public void onSuccess(AgentHandler agentHandler) {
                                try {
                                    Log.i(TAG, "###Success to create agent: " + agentHandler.getAgentController().getName());
                                    exportLogConsole("Success to create agent: " + agentHandler.getAgent(controller().getName());
                                    agentHandler.getAgentController().start();
                                } catch (StaleProxyException e) {
                                    e.printStackTrace();
                                }
                            }
                            @Override
                            public void onFailure(Throwable throwable) {
                                Log.i(TAG, "###Failed to created an Agent");
                            }
                        });

                } else {
                    Log.e(TAG, "##Can't get Main-Container to create agent");
                }
            }
```

Fig 4.2

VI. PERFORMANCE ANALYSIS

We have implemented the framework using a mobile application to evaluate the performance of the framework in terms of memory usage and CPU usage of the mobile device. In our application we have carried out a form of image processing on cloud. We send an image along with a text to Firebase for computation. The image is processed on cloud in a way that the text gets hidden within the image and is

sent back to the device. Thus the computational part is offloaded by the application to the cloud. Our application is based on the concept of covered writing, hiding of a message within another so that the presence of the hidden message is indiscernible which ensures that the people who are not intended to be the recipients of the message should not even suspect that a hidden message exists. The performance analysis was done keeping 2 scenarios in mind i.e. before the network connection to the cloud is established and after the network connection to the cloud is established.

Fig.5.1

Figure 5.1 shows the memory usage of the application against time in seconds before connecting to the cloud.

Fig.5.2

Figure 5.2 shows the memory usage of the application against time in seconds after connecting to the cloud. We see that once the application is connected to the cloud, the memory usage decreases.

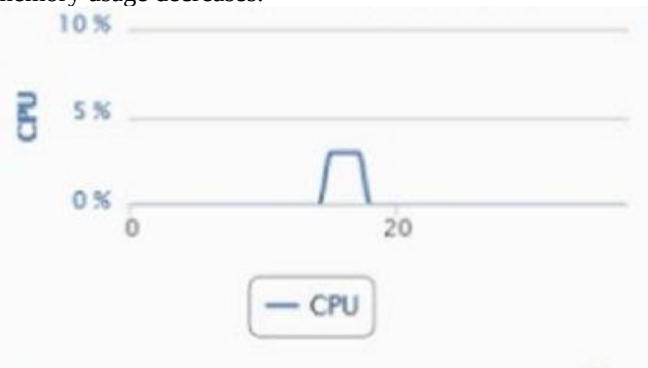

Fig.5.3

Figure 5.3 shows the CPU usage of the application in percentage against time in seconds before connecting to the cloud.

Fig.5.4

The Figure 5.4 shows the CPU usage of the application against time in seconds after connecting to the cloud. There is a rise and dip in the CPU usage once connected to the cloud. Thus we observe that offloading always consumes comparatively less energy and memory than the device-only approach.

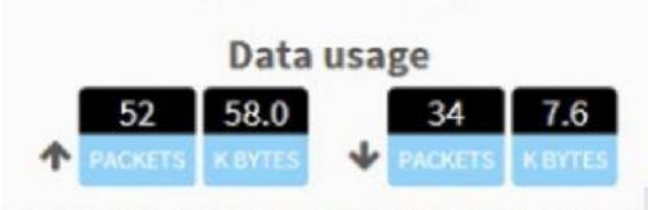

Fig.5.5

The Figure 5.5 shows the monitoring results for the live data usage. It shows the incoming and outgoing packets.

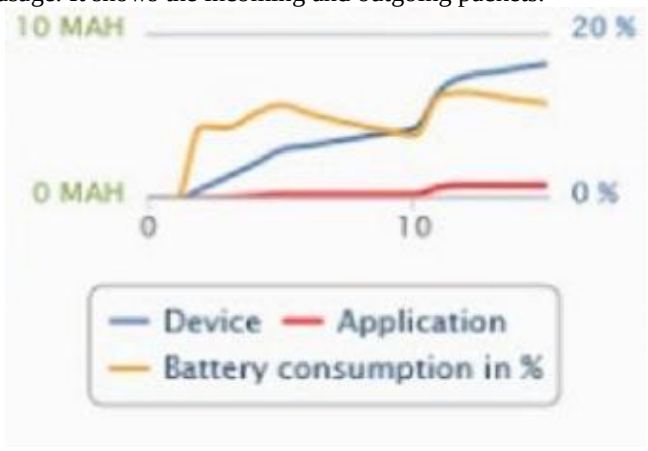

Fig.5.6

The Figure 5.6 shows the battery usage of the device before and after connecting to the cloud, battery usage of the application and the total battery consumption in percentage against the time in seconds. We can see that the battery usage of the application is minimalistic as compared to that of the We have proposed a dynamic performance optimization framework for mobile-cloud computing using mobile agent based application partitions, imposing minimal structural requirements on the cloud. Our approach does not impose any requirements on the cloud platform other than providing isolated execution containers, and it alleviates the management burden of offloaded code by the mobile platform using stateful, autonomous application partitions.

We also investigate the effects of different cloud runtime environment conditions on the performance of mobile-cloud

computing, and present a simple and low-overhead dynamic makespan estimation model integrated into autonomous agents to enhance them with self-performance evaluation in addition to self-cloning capabilities. The proposed performance profiling model is used in conjunction with a cloud resource optimization scheme to ensure optimal performance. Anybody using the trending heavy applications can benefit from this framework. Upcoming virtual reality apps could make use of this framework as they are extremely computation intensive. Many users report substantial battery drain after an update to such VR apps. Users report that the CPU in their phones is staying cranked up, preventing the device from going to sleep. This really burns through the battery. The proposed framework is promising for improved performance and wide adoption in mobile-cloud computing.

VII. FUTURE WORK

Our future work will mainly focus on the following 2 points:

Dynamic partitioning of the agents:
The static partitioning of the agents will fall short in capturing the performance for every possible problem size, as it could prove hard to enumerate all possible problem sizes for a specific program partition.
Our future work will focus on working on a model where the agents are partitioned dynamically so as to optimize the performance.

Security:
One of the main problems in integrating an application with cloud is its security. The security can be easily breached as the computation is carried out on a remote platform. Lack of control over these remote resources may increase the security risks involved. An additional area of concern here is that the agents can be tampered with while on the move.
Our future work will involve working towards reducing these security risks and ensuring secure commutation of the agents to the cloud platform.

APPENDIX

Google Firebase: Firebase is a mobile and web application platform with tools and infrastructure designed to help developers build high-quality apps.
JADE: JADE is a distributed agents platform, which has a container for each host where you are running the agents. Additionally the platform has various debugging tools, mobility of code and content agents, the possibility of parallel execution of the behavior of agents, as well as support for the definition of languages and ontologies.
Mobile Agents: Mobile agent is a process that can transport its state from one environment to another, with its data intact, and be capable of performing appropriately in the new environment. Mobile agents decide when and where to move.
Offloading: Offloading refers to the transfer of certain computing tasks to an external platform, such as a cluster, grid, or a cloud.
Cloud Functions: Cloud Functions is a hosted, private, and scalable Node.js environment where you can run JavaScript code. Firebase SDK for Cloud Functions integrates the Firebase platform by letting you write code that responds to events and invokes functionality exposed by other Firebase features. Function Point Analysis Based Method: The Function Point Analysis is another method of quantifying the size and complexity of a software system in terms of the functions that the systems delivers to the user. A number of proprietary models for cost estimation have adopted a function point type of approach, such as ESTIMACS and SPQR/20.

REFERENCES

[1] Andrea. Colangelo, "Google Cloud vs AWS: A Comparison | Cloud Academy, " Cloud Academy Blog, 20-Feb-2017. [Online]. Available: http://cloudacademy.com/blog/google-cloud-vs-aws-a-comparison/ [Accessed: 17-Aug-2016].

[2] "What Can I Do with Cloud Functions? | Firebase, " Google. [Online]. Available: https://firebase.google.com/docs/functions/use-cases. [Accessed: 22-Sep-2016].

[3] "Running, Testing, and Deploying the Backend | Cloud Tools for Android Studio |Google Cloud Platform," Google. [Online]. Available: https://cloud.google.com/tools/android-studio/app_engine/run_test_deploy. [Accessed: 17-Oct-2016].

[4] P. Angin and B. Bhargava, "An Agent-based Optimization Framework for Mobile-Cloud Computing," Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, vol. 4, no. 2, pp. 1–17, 2013

[5] R. Irani, "Announcing .. Gradle Tutorial Series – Romin Irani's Blog," RominIrani'sBlog,28-Jul-2014.[Online].Available: https://rominirani.com/announcing-gradle-tutorial-series- 5fd134223bf8#.qwoy9n1fi.[Accessed: 02-Jan-2017].

[6] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud In Proceedings of the 6th European Conference on Computer Systems (EuroSys 2011), April 2011.

[7] "Step 4: Create a Mobile Application for Android," Step 4: Create a Mobile Application for Android - AWS Lambda. [Online]. Available: http://docs.aws.amazon.com/lambda/latest/dg/with-ondemand-android-mobile-create-app.html. [Accessed: 09-Jan-2017].

[8] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, RanveerChandra, and Paramvir Bahl; MAUI: making smartphones last longer with code offload. In MobiSys '10:Proceedings of the 8th international conference on Mobile systems, applications, and services, ACM, 2010.

[9] S. Park, Y. Choi, Q. Chen, and H. Y. Yeom, Some: Selective offloading for a mobile computing environment, International Conference on Cluster Computing (CLUSTER12),, in Proc. of the IEEE,

Beijing, China. IEEE, September 2012, pp. 588-591.

[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. ThinkAir:Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in Proc. of the 31st IEEE International Conference on Computer Communications (INFOCOM12), Orlando, Florida, USA. IEEE, March 2012, pp. 945-953.

