An Enhanced Version of the MCACC to Augment the Computing Capabilities of Mobile Devices Using Cloud Computing

Mostafa A. Elgendy
Computer Science
Faculty of Computers and Informatics
Benha, Egypt

Ahmed Shawish
Scientific Computing
Ain Shams University
Cairo, Egypt

Mahmoud I. Moussa
Computer Science
Faculty of Computers and Informatics
Benha, Egypt

***Abstract*—Recently as smartphones have a wide range of capabilities a lot of heavy applications like gaming, video editing, and face recognition are now available. However, this kind of applications need intensive computational power, memory, and battery. A lot of researches solve this problem by offloading applications to run on the Cloud due to its intensive storage and computation resources. Later, some techniques chooses to offload part of the applications while leaving the rest to be processed on the smartphone based on one or two metrics like power and CPU consumption only without any consideration to other important metrics. Our previously proposed MCACC framework has introduced a new generation of offloading frameworks that handle this problem by smartly emerging a group of real-time metrics like total execution time, energy consumption, remaining battery, memory, and security into the offloading decision. In this paper, we introduce an enhanced version of the MCACC framework that can now smartly operate under low bandwidth network scenario in addition to its existing capabilities. In this framework, any mobile application is divided into a group of services, and then each of them is either executed locally on the mobile or remotely on the Cloud through a dynamic offloading decision model. The extensive simulation studies show that both heavy and light applications can benefit from the proposed framework while saving energy and improving performance compare to previous counterparts. The enhanced MCACC turns the smartphones to be smarter as the offloading decision is taken without any user interference.**

Keywords—smartphones; android; offloading; mobile Cloud computing; battery; security

I. INTRODUCTION

Recently smartphones are becoming popular. Studies showed that more than 56% of users in the world use smartphone [5]. More than 53% percent of smartphone owners used the Android OS [6]. Smartphones have a wide range of capabilities like, Wi-Fi, cameras, storage, GPS and speed processors. As a result, developers are building more complex mobile applications such as into heavy applications such as natural language translators, speech recognizers, optical character recognizers, image processors and search, online games, video processing and editing, navigation, face recognition and augmented reality.

As applications become more complex, it consumes most of the mobile devices resources such as battery, memory, and computational power. Mobile applications can augment their capabilities with unlimited computing power and storage space by offloading some services to run on the Cloud; as result saving time and computation power which are called *mobile Cloud computing* [1 - 3].

A mobile Cloud computing survey show a lot of work done in this field. Some solutions considered an application as a single unit that cannot be decomposed into multiple methods and must be run either on the Cloud or locally [9], while others like [7] always offload services to execute on the Cloud all the time without taking any decisions as in. Some other solutions use a simple offloading model which take parameters like power and CPU consumption in their offloading decision as in [8-10]. Later, our proposed Mobile Capabilities Augmentation using Cloud Computing (MCACC) framework [11] has introduced a new generation of offloading frameworks that handle this problem by smartly emerging a group of real-time metrics like total execution time, energy consumption, remaining battery, memory, and security into the offloading decision. Its extensive simulation studies showed its capability to handle heavy applications by efficiently utilize the available smartphone resources and offload only when necessary based on realistic decision metrics.

In this paper, we introduce an enhanced version of the MCACC framework that can now smartly operate under low bandwidth network scenario in addition to its existing capabilities. In this framework, any mobile application is divided into a group of services, and then each of them is either executed locally on the mobile or remotely on the Cloud based on a dynamic offloading decision model. In case of low bandwidth scenario, the offloading decision is taken based on real-time comparisons between being executed locally, or compressed and then offloaded, or offloaded directly without compression.

The extensive simulation studies show that both heavy and light applications can benefit from the proposed framework even under low bandwidth scenario, while saving energy and improving performance compare to its previous counterparts.

Now, Android developers can use our proposed MCACC very easily by adding the MCACC library into their projects and by adding the MCACC builders to the project building process.

In this paper, we provide a detailed description of the whole MCACC framework with its new enhancement as a complete solution. The rest of the paper organized as follows. Section II introduces the background and shows related work. Section III describes the MCACC framework. Section IV discusses the results of the extensive simulation studies. The paper is finally concluded and future work is presented in Section V.

II. BACKGROUD

This section provides a comprehensive background on the mobile environment and application development process. In addition, it provides to a complete review on the related work done in the offloading context.

A. Mobile environment and Application Development

1) Android Architecture: The Android platform is a software stack that was designed primarily but not exclusively to support mobile devices such as phones and tablets. This stack has several layers going all the way from low level operating system services that manage the device itself up to sample applications, things like the phone dialer, the context database, and a web browser. At the bottom, there is a i) Linux kernel layer- which provides two core services that any Android computing device will rely on. The first service is Generic operating system services which contains device drivers, memory management, process management and security. The second service is Android specific components which contains power management, Android shared memory and Inter Process Communication (IPC). Above that, there are ii) System libraries- These libraries are typically written in C and C++ and for that reason they are often referred to as the native libraries. These native libraries handle a lot of the core, performance sensitive activities on your device like quickly rendering web pages and updating the display. Beside this there is iii) Android runtime system – contains two components which support writing and running Android applications. The first component is a core Java libraries that provides a number of reusable Java building block to allow developer to write Android applications using Java programming language. The second component is the Dalvik Virtual Machine that actually executes Android applications. Above that, there's a rich iv) Application framework layer- this exposes the various capabilities of the Android OS for application developers so that they can use these capabilities in their applications. These capabilities are like package manager, window manager, view system, resource manager, activity manager, Content providers, location manager, and notification Manager. Finally at the very top, there is v) Applications - Android comes with some built-in applications which include things like the Home Screen, the Phone Dialer, the Web Browser, an Email Reader, and more. One of the things that are really nice about Android is that none of these apps is hardcoded into the system [12].

2) Android Application Components: In any android applications the following components make the structure of it, and these component are Activities, Services, Content Providers, and Broadcast Receivers, which have their own specific lifecycle within the system [12]. This study focused on activities and services as the separation between the them form a natural basis for MCACC framework.

3) Android IPC: When user launch an android application the operating system starts an activity that presents a graphical user interface to the user. When this activity is bound to the running service, it communicates with the service through IPC, using a predefined interface by the programmer called AIDL file and a stub/proxy pair generated by the Android pre-compiler [13]. When an activity tries to call service method, it uses the proxy object to communicate with the stub which has the actual implementation of the service as shown in Fig. 1.

4) Android Application Development: Any android applications have to be written in the Java. When developer writes android applications and try to build it, the build process will invoke Android Resource Manager followed by Android Pre Compiler, then it invoke Java Builder, finally invoking Package Builder to build a single APK file which can be installed on any Android device [11].

B. Related Work

A lot of researches have been done on remote execution of mobile applications services on the Cloud to increase performance and save mobile power and memory resources [19] and [20]. These researches are divided into two paths:

1) Process and VM Migration: In this approach a full process or full VM is migrated into the Cloud for processing. There are some researches done in this approach as follows:

CloneCloud enables unmodified mobile applications running in an application level virtual machine to seamlessly offload part of their execution from mobile devices on device clones operating in a computational Cloud[14]. When running a complete clone of the smartphone at the remote Cloud resource, there is cost of keeping the smartphone synchronized with an application clone in the Cloud; so it's better to offload only the needed services to run on the Cloud, Also in low bandwidth network data can be compressed before offloading to the Cloud to minimize data transferred over network. ***ThinkAir*** exploits the concept of smartphone virtualization in the Cloud and provides method-level computation offloading [15]. *ThinkAir* creates virtual machines (VMs) of a complete smartphone system on the Cloud, and provides an online method-level offloading however it lacks flexibility and control over offloaded components. Developers organize their application using Android service design patterns. Also in low bandwidth network data can be compressed before offloading to the Cloud to minimize data transferred over network.

2) Method Offloading: Another common approach for remote execution is to partition mobile application into some

services that executes locally on mobile or remotely on the Cloud and this is called method offloading. There are a lot of researches which have done in this approach and these will be described and their drawbacks will be discussed in the following section.

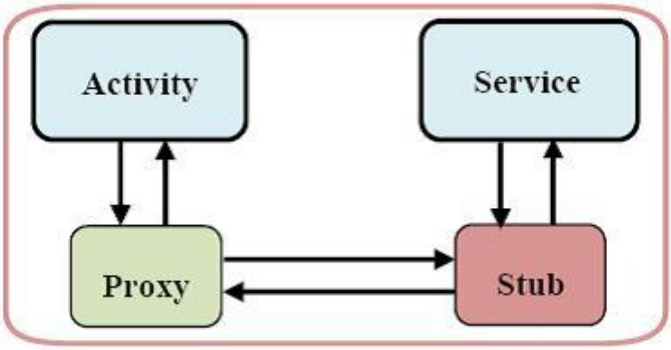

Fig.1. An overview of the Android IPC mechanism

Cuckoo proposed a framework that automatically offloads heavy services to execute on the Cloud. *Cuckoo* use the very simple model which always prefers remote execution [7]. It's better to use some metrics in taking offloading decision like service processing time instead of offloading all the time than always offloading service directly. Also, in low bandwidth networks the time for communicating and transmit data on network and execute service on the Cloud is larger than the time to executing services on mobile, so *Cuckoo* decide to run services locally on the mobile. *Cuckoo* can compress data before sending to the Cloud, as result saving time in low bandwidth networks. Another thing, data sent to Cloud must be protected, so any security technique should be applied for protection. ***Eric Chen*** implements a framework that automatically offloads heavy tasks to execute on the Cloud [8]. *Eric Chen* uses total response time, energy consumption and remaining battery life in deciding whether a task should be offloaded or not without adding any memory usage consideration and security to the offloading model. Also in low bandwidth networks, application can also offload data to the Cloud by compressing data before offloading which lead to minimize data transferred over network. ***Vinod*** proposed a model for deciding whether to offloads heavy backend tasks to execute on the Cloud [9]. *Vinod* take some considerations like memory usage consideration and security to the offloading model. *Vinod* considered an application as a single unit that cannot be decomposed into multiple methods and must be run either on the Cloud or locally. However in some cases it's better to offload some methods to execute on the Cloud and run the others on mobile. ***Karthik Kumar*** provides simple analysis for deciding whether to offload computation to a server or not. This analysis tries to measure the power of sending computations to the Cloud and the power of executing computation on mobile device [1]. Although this analysis solved the problem, it lack any memory usage consideration and battery consideration when making analysis. *Kumar* also conclude that offloading data intensive tasks to the Cloud depends on the network bandwidth as if the network is low, it will better to execute service locally on the mobile and if the network is high, it will better to execute service remotely on the Cloud. However in low bandwidth networks application may get rid of Cloud by compressing data before offloading, as result execution time and power consumption can be save ***Kiran I. Koshy*** try to measure energy benefits of offloading tasks from mobile devices to powerful remote servers. *Kiran* measured the energy consumed by mobile and added network energy consumed to it, and measure the energy consumed by Cloud and compared for deciding whether a task offloading reduced energy or not [10]. *Kiran* missed some metrics when making this investigation like memory usage. Another thing, data sent to Cloud must be protected, so any security technique should be applied for protection. *Kiran* can be improved by compressing data before offloading to the Cloud in low bandwidth networks. ***Phone2Cloud*** [21] use a naive history based method to predict average execution time of an application on smartphone. It monitor network bandwidth and leverages average CPU workload got from the resource monitor and input size of the application to predict execution time using the history log. However in data intensive application and low bandwidth network, *Phone2Cloud* always prefer to run service locally on the mobile. *Phone2Cloud* can improve his framework by compressing data before offloading to the Cloud in low bandwidth networks.

III. PROPOSED FRAMEWORK DESIGN

In this section, the enhanced MCACC architecture is addressed in detail with its dynamic offloading model. The process done on the Cloud side and the communication between the mobile and the cloud is also discussed. Finally, this section describes the builders added to allow any android application to make use of our framework.

A. MCACC Architecture

As shown in Fig. 2. MCACC consists of four main components *i) Decision Manager - ii) Offloading Manager - iii) Execution profile - and iv) Cloud Manager*. The first three components are deployed on the mobile and the *Cloud manager* component is deployed on the Cloud. In order to use MCACC, the application should be structured using android services pattern. Note that Communication between activities and services done through stub/proxy generated by Android pre-compiler.

*1) Offloading Manager**:** is responsible for executing the application services based on the decision taken by Decision Manager. If the decision is to execute the service locally on the mobile, then Offloading Manager calls the local service implementation from the mobile side. However if the decision is offloading the service for execution on the Cloud, then the Offloading Manager connect to the Cloud Manager and send any data needed to execute the service, Then it waits until the Cloud Manager execute the service on the Cloud and send the result back to the mobile side. At the end Offloading Manager is responsible for receiving the returned results and delivering it to the application.*

*2) Execution Profiler**:**is a profile created for each service by Decision Manager at the first of its run to store some data related to each service like execution time, power comsumption and memory consumption. It store these data for*

each service based on executing sample example of the service in the following three scenarios. The first scenario is when service executed locally on the mobile. The second scenario is when service offloaded and executed on the Cloud. Note in this case it store only Cloud execution time removing the time to send data over network, as the time of sending data over network depends on network bandwidth, so it can be calculated when trying to call the service. The third scenario is when compressing service data, then offloading it for execution on the Cloud. In subsequent runs of the Decision Manager, data stored in Execution Profiler and network bandwidth will be used in taking the offloading decision.

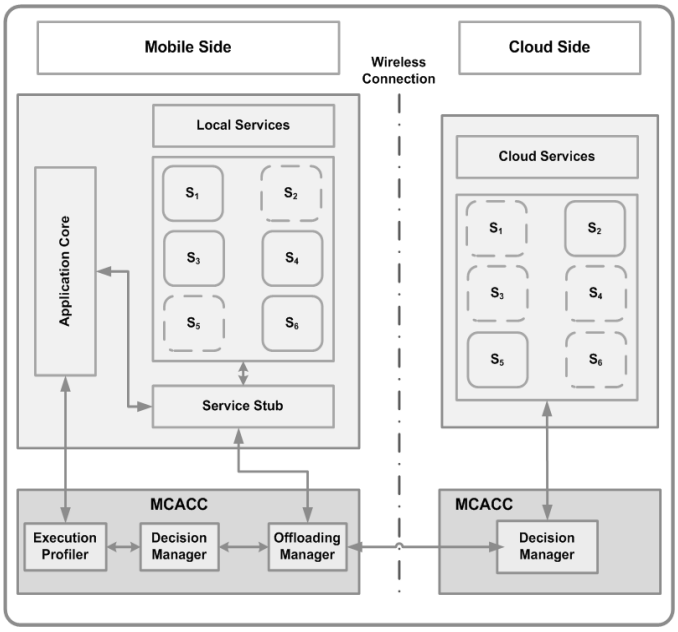

Fig.2. MCACC Architecture – services like S2 can be offloaded to run on the Cloud using Offloading Manager and Cloud Manager.

3) Decision Manager: uses a dynamic offloading model to decide at runtime weather the service will be offloaded to the Cloud – offloaded with or without compression, or executed locally on the mobile. First it get the network bandwidth, then it read service stored data about execution time, power comsumption and memory consumption from the execution profiler for the three running scenarios. Finally it uses the offloading model algorithm described in Section B to make an offloading decision. When it decides to run the application locally or remotely, it calls Offloading Manager which is reasonable for service execution.

4) Cloud Manager: is reasonable for service execution on the Cloud. In the first run it receives the Jar file which contains the remote implementation of all application services and the needed libraries from Offloading Manager and install it at the Cloud side. At any time when the Offloading Manager try to call service from the Cloud, Cloud Manager receive all required data to execute the service, execute it and return the result to Offloading Manager.

B. Offloading Model

When an activity invokes a method of a service, the Android IPC mechanism directs this call through the proxy and the kernel to the stub. In normal android application the stub invokes the local implementation of the method and then returns the result to the proxy. When using the MCACC, the android application becomes smarter. MCACC uses dynamic offloading model to evaluate whether it is beneficial to offload the method to run on the Cloud, compressing it and then offloading to run on the Cloud, or executing it locally on mobile. MCACC uses five metrics in taking decision. These metrics are *i)* execution time- *ii)* energy consumption- *iii)* remaining battery life - *iv)* memory usage and *v)* security.

The ***execution time metric***, dealt with total time required to perform a task. Let *I* be the number of instructions involved in a method invocation, S_{Mobile} be the processor speed (instructions per second) of the mobile and S_{Cloud} be the processor speed on the Cloud. If the amount of data transferred between mobile and Cloud is *D* and the network bandwidth is *B*, the time it takes to transfer data is *D/B*. Using these, MCACC derived the relationship between execution speed and communication overhead as shown:

Let $T_{Mobile} = \frac{I}{S_{Mobile}}$, $T_{Cloud} = \frac{I}{S_{Cloud}}$ and $T_{Net} = \frac{D}{B}$ .from these we can drive this

If $T_{Mobile} - (T_{Net} + T_{Cloud}) > 0$
T=1
else If $T_{Mobile} - (T_{Net} + T_{Cloud}) > 0$ after compression
T=1
else
T=0 (1)

If the execution time on the mobile is greater than the sum of the time to send data over network and execution time on the cloud or if this is true but in the case of compressing data before sending to the Cloud, then it's beneficial to offload to run on the Cloud. Other than theses case it's better to execute service locally on the mobile.

The ***energy consumption metric***, dealt with energy consumption. Let P_{Mobile} *watt* the energy consumed by mobile for computing per second, P_{Cloud} *watt* the energy consumed by mobile for being idle until executing service on the cloud per second and P_{Net} *watt* for sending and receiving data; then the energy consumed is $P_{MTot} = P_{Mobile} \times T_{Mobile}$ *watt*. If the Cloud performs the computation, the energy consumed was $P_{NTot} = P_{Net} \times T_{Net}$ *watt* for the communication overhead and $P_{CTot} = P_{Cloud} \times T_{Cloud}$ *watt*. Using these MCACC derived the relationship between energy consumption on the mobile and on the Cloud:

If $P_{MTot} - (P_{NTot} + P_{CTot}) > 0$
P =1
else if $P_{MTot} - (P_{NTot} + P_{CTot}) > 0$ after compression
P =1
else
p =0 (2)

If the energy consumption on the mobile is greater than the sum of the energy consumed to send data over network and execution time on the cloud or if this is true but in the case of compressing data before sending to the Cloud, then it's beneficial to offload to run on the Cloud. Other than theses case it's better to execute service locally on the mobile.

The ***remaining battery metric***, dealt with mobile remaining battery life in decision-making. Let *L* be the mobile remaining battery life in *watt*. If a task couldn't be completed with the remaining battery of the mobile or if the remaining battery is sufficient to upload the input data required to perform the task on the Cloud, the Cloud can do the task while the mobile's battery drains out. The Cloud can later return the results to mobile. This condition expressed by the following metric:

$$\begin{aligned} &\textit{If } \; L - (P_{NTot} + P_{CTot}) > 0 \\ &\quad B = 1 \\ &\textit{else if } L - (P_{NTot} + P_{CTot}) > 0 \text{ after compression} \\ &\quad B = 1 \\ &\textit{else} \\ &\quad B = 0 \end{aligned} \tag{3}$$

$$\begin{aligned} &\textit{If } \; P_{MTot} - L > 0 \\ &\quad B = 1 \\ &\textit{else} \\ &\quad B = B \quad \textit{- from previous equation} \end{aligned} \tag{4}$$

The ***memory usage metric***, dealt with memory used to perform a task. Let mem_{avail} the memory available on mobile, mem_{total} the total memory available and mem_{th} is the percentage of threshold that the process will not exceed. If the service memory usage exceeded the threshold specified with or without compression, the service offloaded for execution on the Cloud and return the results to mobile, otherwise the service will be executed locally on the mobile. This condition expressed by the following metric:

$$mem_{calc} = \frac{mem_{avail}}{mem_{total}} \times 100$$

From the above equation

$$\begin{aligned} &\textit{If } \; mem_{th} - mem_{calc} > 0 \\ &\quad M = 1 \\ &\textit{else if } \; mem_{th} - mem_{calc} > 0 \quad \text{after compression} \\ &\quad M = 1 \\ &\textit{else} \\ &\quad M = 0 \end{aligned} \tag{5}$$

The ***security metric*** that MCACC use when we try to offload dealt with security used; is the user needs a security on data before sending to the Cloud, using this metric allows to encrypt data before sending and decrypt it on the Cloud for processing. MCACC uses ***AES*** technique for encryption and decryption.

$$\begin{aligned} &\textit{If } \text{ user need security} \\ &\quad S = 0 \\ &\textit{else} \\ &\quad S = 1 \end{aligned} \tag{6}$$

After calculating *T*, *P*, *B*, *M* and *S* from these previous metrics mobile user can set priorities to each metric using the following weights w_t, w_P, w_l, and w_m, finally the offloading model decide whether the service will be offloaded to the Cloud – offloaded with or without compression, or executed locally on the mobile using the following equation:

$$\begin{aligned} &\textit{Let } \; C = T \times w_t + P \times w_p + B \times w_l + M \times w_m \\ &\quad \text{If } C > 0.5 \\ &\quad\quad \textit{Cloud} = 1. \\ &\quad \text{else} \\ &\quad\quad \textit{Cloud} = 0. \end{aligned} \tag{7}$$

After calculating *C* from the previous equation and selecting whether needing security or not. If *C* is greater than 0.5, then the service will be executed on the Cloud, otherwise the services will be executed locally on the mobile. When offloading the service for execution on the Cloud if the user select security, then data will be encrypted before sending it to the Cloud.

C. Cloud Side

Cloud Manager is written with pure Java so any application can offload its computation to any resource running a Java Virtual machine; either being machines in a commercial Cloud such as Amazon EC2 [17] or private Clouds such as laptops and desktops. MCACC run *Cloud Manager* which handles all offloading requests from the clients, installation of offloaded services and their initialization, libraries needed and. Finally *Cloud Manager* invokes services when *Offloading Manager* needs to call them. Note that at first run of user application MCACC sends the jar file created by *Jar Creator* to the Cloud.so all mobile services become available for execution on the Cloud.

D. Communication: IBIS

In order to execute methods on a remote resource, the phone has to communicate with the Cloud resource. MCACC used the Ibis communication middleware for this purpose [11]. The Ibis middleware consists of two subsystems, the Ibis Distributed Deployment System and the Ibis High-Performance Programming System. MCACC framework has been implemented on top of the Ibis High Performance Programming System, which offered an interface for distributed applications [16].

E. Integration into Build Process

In any Android application the connection between the activities and AIDL services processed as follow: When an activity needs to invoke a method in a service, it makes call to the matching method in the proxy. The proxy is responsible for connecting to service to call the need method. The proxy doesn't connect to service directly but, it connects to stub which call the local service and return the result to the proxy. The proxy takes this result and passes it to the caller activity. The framework is deployed in the application layer without modifying the underlying Android platform. The framework

provided three Eclipse builders that can be inserted into an Android project's build configuration in Eclipse.

1) Stub Modifier: The first builder is called the Stub Modifier and has to be invoked after the Android Pre Compiler, but before the Java Builder. The Stub Modifier will rewrite the generated Stub for each AIDL interface, so that at runtime it connected to the Decision Manager to take offloading decision whether a method will be invoked locally on mobile or remotely on the Cloud.

2) Remote Creator: The second builder called Remote Creator used to derive a dummy remote implementation from the available AIDL interface for each service. Now the application with two copies of a service during the build process: i) the first copy of the service added by Android called the local service that executes on the mobile. -ii) the second copy of the service added by framework using Remote Creator and contains the same implementation as the local services and called remote service. This second copy will be executed on the Cloud, so developer can change its implementation to use all Cloud resources like parallel processing.

3) Jar Creator: The third builder called Jar Creator used to build a Java Archive File (jar) which contains the remote implementation and all needed libraries. This jar file will be installed on the Cloud. The Remote Creator and the Jar Creator have to be invoked after the Java Builder, but before the Package Builder, so that the jar will be part of the Android Package file that results from the build process as shown in Fig. 3.

IV. Simulation Studies

To evaluate the MCACC framework, a face detection application was used. It is an application that allow user to select image from gallery or to take real-time one, then the application execute face detection service locally on mobile or remotely on the cloud using the proposed enhanced MCACC framework. After that detection service return an array of all detected faces. Finally the application use this array to draws a rectangle around each detected face as shown in Fig.4. This application uses *JavaCV* library to detect image faces. *JavaCV* is a wrapper that allows accessing the *OpenCV* library directly from within Java Virtual Machine (JVM) and Android platform.

A. Simulation Setup

Hardware: On the mobile side a Samsung Galaxy S Advance GT-I9070 mobile was used. The mobile uses Android operating system in version 4.1.2, integrates with Wi-Fi interface, and a battery capacity of 1500mAh. It has CPU with 1 GHz, 1.97 GB system storage and 3.92 GB USB storage at 3.7 volts. On the Cloud side a laptop with a core I3 2.13 processor, 4 Giga Ram acted as a Cloud provider. We evaluate the execution time, power consumption and CPU consumption for our application. To measure the power consumption, CPU consumption, and used memory a software called little eye V2.4.0.0 is used [18].

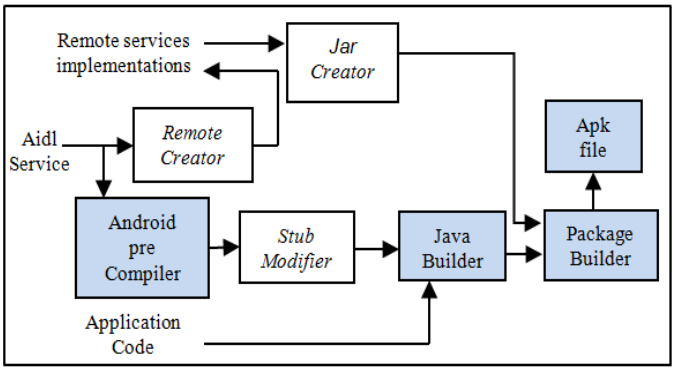

Fig.3. A schematic overview of how components integrate into the default build process.

B. Result and Discussion

Five images were used in the evaluation of the face detection application. The application was evaluated three times. *i)* First the application was evaluated in a good bandwidth network under two scenarios; the first one represents the execution of the face detection service on the mobile device, while the second one represents the offloading of the service for execution on the Cloud. *ii)* Second the application was evaluated in a low bandwidth network under three scenarios; the first one represents the execution of the face detection service on the mobile device, the second represents the offloading on the Cloud and the third represents compressing the data before offloading it on the Cloud. and finally *iii)* the application was evaluated using more than on security algorithms under two scenarios; the first one represents the execution on the mobile device, while the second one represents the offloading on the Cloud.

Fig.4. Screenshot of face detection application

1) Good bandwidth network: Five images were used in evaluating the face detection application under two scenarios; the first one represents the execution of the face detection service on the mobile device, while the second one represents the offloading of the service for execution on the Cloud.

Fig. 5 shows the execution time where the x -axis represents the size of the images in kilo bytes while the y-axis represents the processing time in seconds. It can be easily noted that the increase of image's size implies a corresponding increases in the processing time on the mobile device consuming memory and power resources while in the offload scenario (i.e., using the Cloud) such resources are relatively preserved.

For example, the image with size 9830.4 kb takes about 7 seconds to be executed on mobile while it takes about 5 second to be executed on Cloud. However, it is worth to note that the time in the offloading scenario Cloud is the sum of the time needed to send/receive the service to/from the Cloud plus the execution time there. So the offloading scenario does not only depends on the Cloud execution time but also depends on the network bandwidth.

Fig.6. shows the CPU consumption percentage in both scenarios. The x -axis represents the size of images in kilo bytes and the y-axis represents the average of CPU consumption percentage. The result demonstrates the aggressive consumption of the mobile resources in case of executing such heavy service. It also shows the efficiency of the offloading approach to save such resources. For example, the execution of face detection service on mobile consumed about 33% of CPU, while this percentage is minimized to 7.5% in the offloading scenario.

Fig. 7 and 8. describe the power and memory consumption in both scenarios, respectively. The x -axis represents the size of images in kilo bytes and y-axis represents the power and memory consumed by the mobile. The results of both experiments match well with the conclusion of the previous one: offloading is a better choice in case of heavy services. However, we are not arguing to prove this conclusion, we are here providing a smart offloading framework that is able to take the right decision under any circumstance, taking into consideration all of the above real-time metrics.

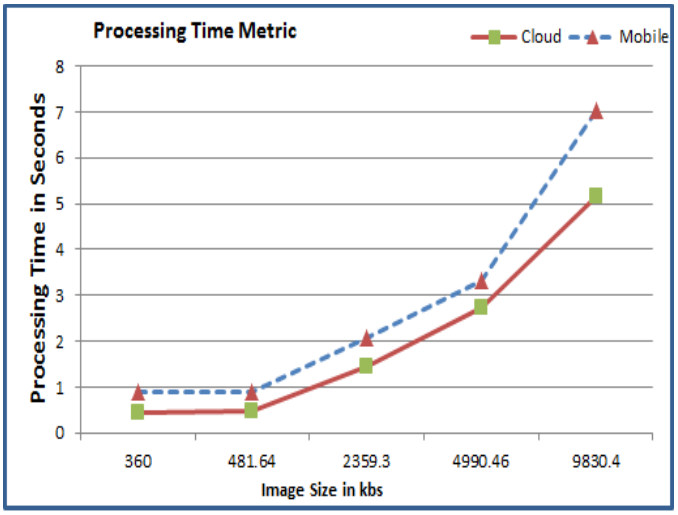

Fig.5. Processing time on mobile and on Cloud

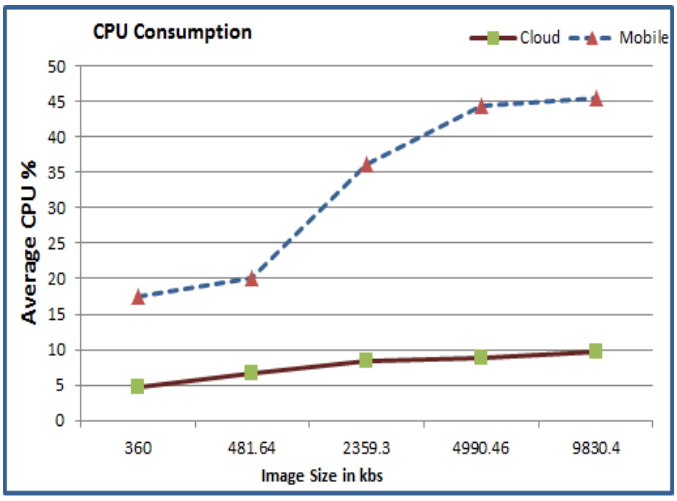

Fig.6. CPU consumption percentage on mobile and on Cloud

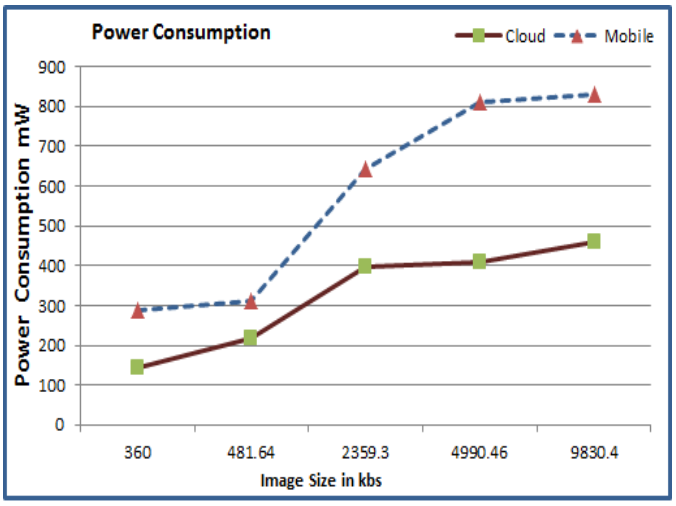

Fig.7. Power consumption on mobile and on Cloud

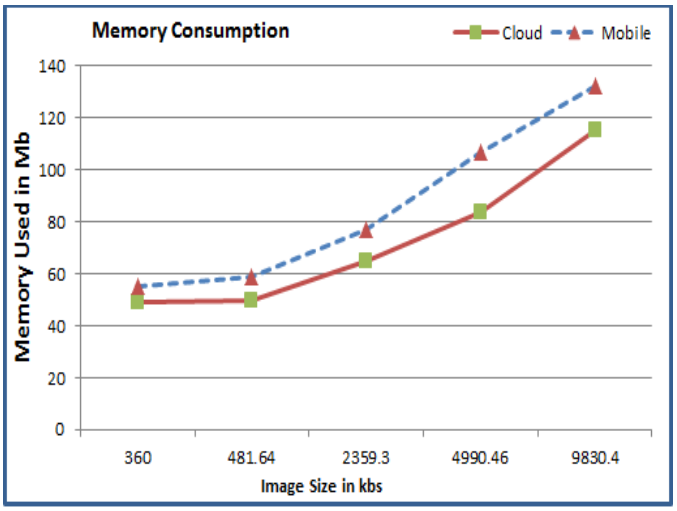

Fig.8. Memory consumption on mobile and on Cloud

1) Low bandwidth network: In this part the application is evaluated in a low bandwidth network under three scenarios; the first one represents the execution of the face detection service on the mobile device, the second represents the offloading on the Cloud, and the third represents compressing the data before offloading.

Fig.9. shows the execution time in the three scenarios in low bandwidth networks. The x -axis represents the size of the images in kilo bytes and the y-axis represents the processing time in seconds. It can be easily noted that the execution time on the Cloud without compression is greater than execution time on mobile and on the Cloud with compression as it takes more time to transfer data through the low bandwidth network.

For example, the image with size 9830.4 kb takes about 12 seconds to be executed on Cloud while it takes about 7 second when executed on the mobile. Similarly, as the image size increases, the execution time on the Cloud without compression increases too in comparison with the other two scenarios. It is also noted that the execution time on the mobile is nearly equal to the compression scenario. For example, the image with size 9830.4 kb takes about 7 second to be executed on the mobile and almost the same when offloaded on the Cloud with compression. Accordingly, we conclude that compressing data and offloading it will give the same performance as processing the requested service on the mobile; nevertheless it will save the mobile resources.

Fig.10. shows the CPU consumption percentage in the three scenarios. The results demonstrate the aggressive consumption of the mobile resources in case of executing such heavy service locally on the mobile. It also shows the efficiency of offloading service to save such resources.

For example an image with size 9830.4 kb consumes about 48% of the mobile CPU in the first scenario while consuming 10% and 16% in the second and third scenarios, respectively. It also noted that the execution of face detection service on mobile consumed about 34% of CPU on average, while this percentage is minimized to 12.45% in the compression offloading scenario and 7.4 % in the offloading scenario without compression. Accordingly, it can be concluded that in low bandwidth networks, if the user priority is to save the mobile CPU consumption, then it is better to offload service to the Cloud with or without compression.

Fig.11. describes the power consumption in the three scenarios, respectively. The results match well with the conclusion of the previous one; offloading data to the Cloud or compressing data and then offloading to the Cloud is a better choice in case of heavy services if the network bandwidth is low. the extensive simulation studies report that in low bandwidth network it is better to compress the data before offloading to the Cloud.

2) Security: the application was evaluated with more than on security algorithms using two scenarios; the first one represents the execution on the mobile device, while the second one represents the offloading on the Cloud.

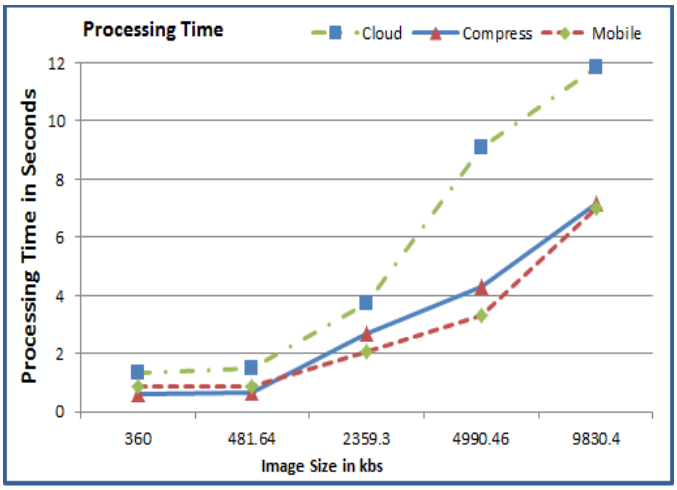

Fig.9. Processing Time of the application under the three scenarios using low bandwidth network

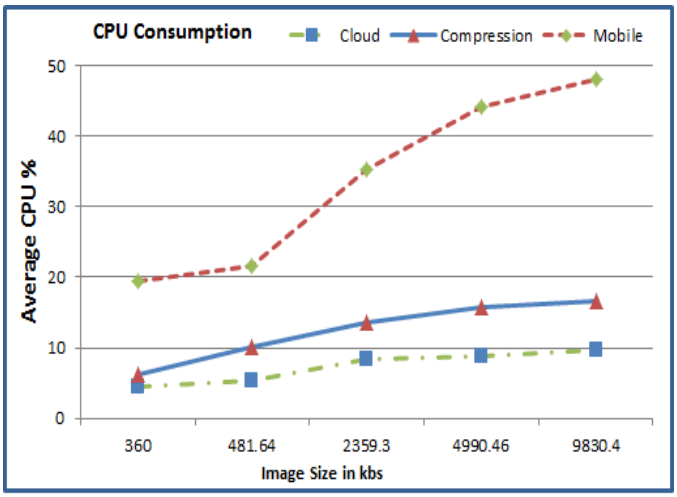

Fig.10. CPU Consumption of the application under the three scenarios using low bandwidth network.

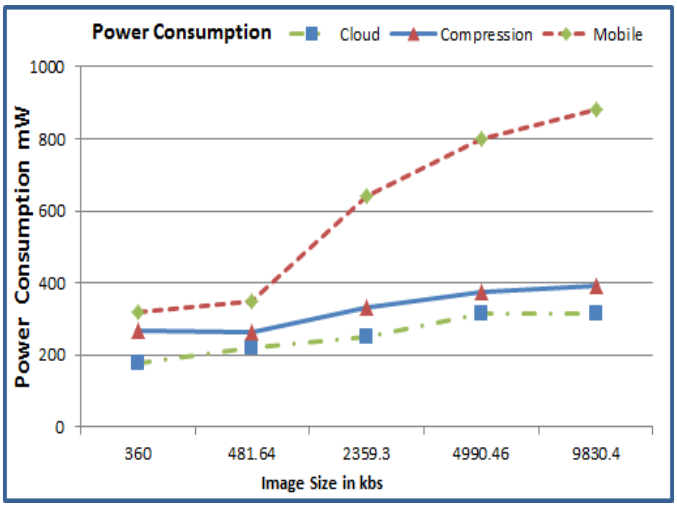

Fig.11. Power Consumption of the application under the three scenarios using low bandwidth network

Fig.12. shows the execution time when running the application on the mobile versus a secured offloading. As usual, the x -axis represents the size of images in kilo bytes and y-axis represents the processing time in millisecond. It should be noted here that before offloading the service (i.e., sending the faces image to be detect on the Cloud) the MCACC will first encrypt it and then decrypt it after receiving the results from the Cloud. The results shows that adopting a security layer on the transmitted data add an overhead on the mobile resources in order to be encrypt and decrypt. In some case this processing time is acceptable in small sized images like for example image of size 360 kb takes (0.583 second) in the secured offloading scenario while it takes (0.885 second) on the mobile. In other case like for example image of size 9830.4 kb, it takes (7.49 seconds) in the secured offloading scenario while it takes (7.012 seconds) on the mobile. It is also worth to note that the AES technique for encryption and decryption is better than the Blowfish technique as shown it the figure. We can easily note the effect of security layer on the offloading process and how it may affect the processing time on the mobile.

In general, the extensive simulation studies report that executing application services on the mobile consumed a lot of the mobile's resources which is not acceptable, while offloading it to the Cloud may save such resources. Also the results showed that in low bandwidth network, application services can be offloaded by compressing data before offloading. The results also showed that when adding a security layer to the offloading process an additional overhead should be taken into consideration. On the other hand, it is worth to note that the proposed framework supports automatic offloading of multiple Android services based on a group of realistic metrics inspected instantaneously from the smartphone. In addition, the with the popular open source Android framework and the Eclipse development tool. It provides a simple programming model, familiar to developers. This model allows developer to use our framework very easily and adds offloading components automatically.

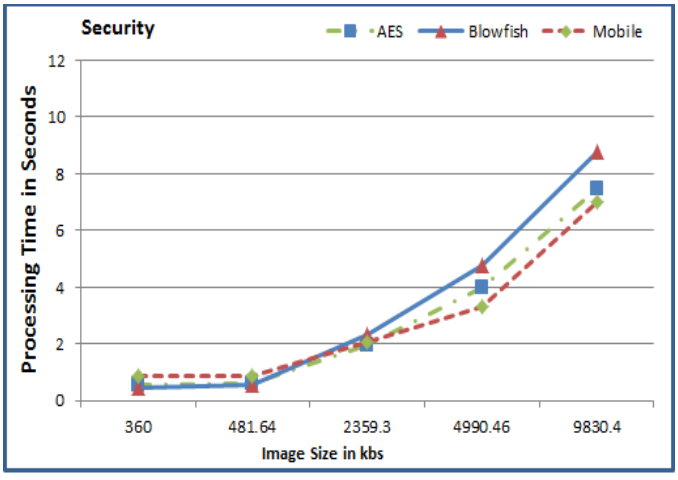

Fig.12. Processing time on mobile and on Cloud when Appling encryption techniques

The Proposed MCACC is efficiently solving a group of drawbacks in the current available techniques. For example, it overcomes the Clone Cloud [14] deficiency by offloading only the needed services based on the offload model, and hence avoids the costly process of keeping the smartphone synchronized with an application clone in the cloud. Moreover, by adopting a runtime offloading model based on five metrics, MCACC is smarter than Cuckoo [7] that uses a very simple heuristic approach to always send services to be executed on cloud without any decision. With respect to the solutions provided by Eric Chen [8], Vinod Namboodiri [9], Karthik Kumar [1], and Kiran I. Koshy [10] that utilize metrics like total response time, energy consumption and battery power in their offloading decision, MCACC is still better as it additionally utilizes the memory and security metrics. Also MCACC use cloud also in low bandwidth network by compressing data before offloading to the Cloud, so saving mobile resources.

V. CONCLUSION

This paper proposed an enhanced version of the framework called Mobile Capabilities Augmentation using Cloud Computing (MCACC) that helps smartphone to handle heavy applications. The new enhancement extends the previous framework capabilities to utilize the limited available resources of the smartphones and smartly offload the services to the Cloud even under low bandwidth scenario. In this framework, any mobile application is divided into a group of services, and then each of these services are either executed locally on the mobile or remotely on the Cloud using a dynamic offloading decision model. Here, the decision is based on real-time metrics: total execution time, energy consumption, remaining battery, memory, security, and network bandwidth.

The extensive simulation studies report the ability of the proposed framework to efficiently utilize the available smartphone's resources in addition to augmenting them using the Cloud Computing.Our future work will focus on enabling parallelization of the offloaded services and minimizing the security overhead between the mobile and the Cloud.

REFERENCES

[1] K. Kumar, and L.Yung-Hsiang, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Computer* , vol.43, no.4, pp.51-56, April 2010 doi: 10.1109/MC.2010.

[2] P. Mell, and G. Timothy, "The NIST definition of Cloud computing," *NIST special publication*, 800(145), 7. 2011.

[3] D. Kovachev, C. Yiwei, and K. Ralf, "Mobile Cloud Computing: A Comparison of Application Models, " *Computing Research Repository*, abs-1107-4940, 2011.

[4] A. Berl, G. Erol, D. G. Marco, G. Giovanni, D. M. Hermann, Q. D. Minh, and P. Kostas, "Energy-Efficient Cloud Computing," *The Computer Journal* vol. 53, pp. 1045-1051. 2010.

[5] webpage on Smartphone users. [Online]. Available: http://www.go-gulf.com/blog/smartphone/. 2012.

[6] Hepburn, A. webpage on Android usage. [Online]. Available: http://www.digitalbuzzblog.com/infographic-2013-mobile-growth-statistics/. 2013.

[7] R. Kemp, P. Nicholas, K. Thilo, and B. Henri, "Cuckoo: A Computation Offloading Framework for Smartphones," *Mobile Computing, Applications, and Services.*, vol. 76, pp. 59-79, 2012.

[8] E. Chen, O. Satoshi, and H. Keitaro, "Offloading Android applications to the Cloud without customizing Android," in *Pervasive Computing and Communications Workshops IEEE International Conference on*, pp. 788-793, 19-23 Mar 2012.

[9] V. Namboodiri, and G. Toolika, "To Cloud or not to Cloud: A mobile device perspective on energy consumption of applications," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pp. 1-9, 25-28 Jun 2012.

[10] K. I. Koshy, M. J. Andrew, N. Andrew, and O. Michael, "Can Cloud computing lead to increased sustainability of mobile devic?," *Sustainable Systems and Technology (ISSST), 2012 IEEE International Symposium on*, pp.1-4, 16-18 May 2012.

[11] M. A. Elgendy, S. Ahmed, and I. M. Mahmoud, "MCACC: New Approach for Augmenting the Computing Capabilities of Mobile Devices with Cloud Computing", *Science and Information Conference (SAI)*, pp 79-86, 27-29 Aug 2014.

[12] M. Gargenta, Learning Android Building Applications for the Android Market, O'Reilly Media, 2011.

[13] Webpage on Android Developer AIDL [Online]. Available: http://developer.android.com/guide/components/aidl.html. 2013.

[14] B. G. Chun, I. Sunghwan, M. Petros, N. Mayur, and P. Ashwin, " CloneCloud: elastic execution between mobile device and Cloud, " *6th conference on Computer systems (EuroSys)*, pp. 301–314, 2011.

[15] S. Kosta, A. Andrius, H. Pan, M. Richard, and Z. Xinwen, "ThinkAir: Dynamic resource allocation and parallel execution in the Cloud for mobile code offloading," *INFOCOM, 2012 Proceedings IEEE*, pp. 945-953, 25-30 Mar 2012.

[16] R. V. V. Nieuwpoort, M. Jason, H. Rutger, K. Thilo, and E. B. Henri, "Ibis: an Efficient Java-based Grid Programming Environment," in *Joint ACM Java Grande - ISCOPE 2002 Conference*, pp. 18-27, 2002.

[17] The Amazon Elastic Computing website. [Online]. Available: http://aws.amazon.com/ec2/. 2013.

[18] The Little eye website. [Online]. Available: http://www.littleeye.co/. 2014.

[19] E. Cuervo, B. Aruna, K. C. Dae, W. Alec, S. Stefan, C. Ranveer, and B. Paramvir, "MAUI: making smartphones last longer with code offload," in *International Conference on Mobile Systems, Applications, and Services*, pp. 49-62, 2010.

[20] M. Shiraz, G. Abdullah, H. K. Rashid, and B. Rajkumar, "A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing," *Communications Surveys & Tutorials, IEEE* , vol.15, no. 3, pp. 1294-1313, Third Quarter 2013.

[21] Xia, F., Ding, F., Li, J., Kong, X., Yang, L.T., Ma, J., "Phone2Cloud Exploiting computa-tion offloading for energy saving on smartphones in mobile Cloud computing", In: Information Systems Frontiers, vol.16, no. 1, pp. 95-111, (2014).